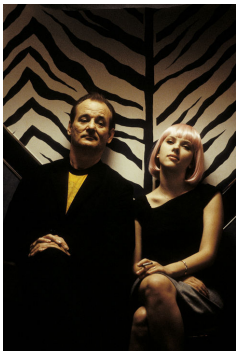


# Lost in translation: Formalizing proposed extensions to C<sup>#</sup>

Gavin Bierman<sup>1</sup>   Erik Meijer<sup>2</sup>   Mads Torgersen<sup>2</sup>



<sup>1</sup>Microsoft Research, Cambridge

<sup>2</sup>Microsoft Corporation

ooPSLA 2007

# LINQ (Language INtegrated Query)

- An approach to addressing the impedance mismatch problem
- Aim to make data programming:
  - Easy
  - Regular
  - Extensible
- Extend C<sup>#</sup> and Visual Basic to make this programming even more pleasant

# LINQ (Language INtegrated Query)

- An approach to addressing the impedance mismatch problem
- Aim to make data programming:
  - Easy
  - Regular
  - Extensible
- Extend C# and Visual Basic to make this programming even more pleasant

**And all this with no changes to the CLR (just some new libraries)**

Demo!

# Aims of “lost in translation”

Subject of study: The new language features

Several different perspectives:

- To provide a concrete semantics for C#3.0
- (Actually to formalize some C#2.0 features for the first time!)
- To show how the new features in C#3.0 can be compiled into existing C#2.0 features
- To demonstrate/prove that no changes to the CLR are needed

# Aims of “lost in translation”

Subject of study: The new language features

Several different perspectives:

- To provide a concrete semantics for C#3.0
- (Actually to formalize some C#2.0 features for the first time!)
- To show how the new features in C#3.0 can be compiled into existing C#2.0 features
- To demonstrate/prove that no changes to the CLR are needed
  - To show how to build a C#3.0 compiler on the cheap (but don't tell the lawyers!)

Two stages:

- 1 Translate query expressions into method invocations
- 2 Translate the remaining C#3.0 code into C#2.0

# Query expression compilation

```
var names =  
  from c in Customers  
  where c.City == "Redmond"  
  where c.Credit > 10000  
  orderby c.Age descending  
  select c.Name;
```

# Query expression compilation

```
var names =  
    from c in Customers  
    where c.City == "Redmond"  
    where c.Credit > 10000  
    orderby c.Age descending  
    select c.Name;
```

```
var names =  
    from c in Customers  
        .Where((c)=>c.City=="Redmond")  
    where c.Credit > 10000  
    orderby c.Age descending  
    select c.Name;
```

# Query expression compilation

```
var names =  
  from c in Customers  
    .Where((c)=>c.City=="Redmond")  
  where c.Credit > 10000  
  orderby c.Age descending  
  select c.Name;
```

# Query expression compilation

```
var names =  
    from c in Customers  
        .Where((c)=>c.City=="Redmond")  
where c.Credit > 10000  
orderby c.Age descending  
select c.Name;
```

```
var names =  
    from c in Customers  
        .Where((c)=>c.City=="Redmond")  
        .Where((c)=>c.Credit>10000)  
orderby c.Age descending  
select c.Name;
```

# Query expression compilation

```
var names =  
  from c in Customers  
    .Where((c)=>c.City=="Redmond")  
    .Where((c)=>c.Credit>10000)  
  orderby c.Age descending  
  select c.Name;
```

# Query expression compilation

```
var names =  
    from c in Customers  
        .Where((c)=>c.City=="Redmond")  
        .Where((c)=>c.Credit>10000)  
    orderby c.Age descending  
    select c.Name;
```

```
var names =  
    from c in Customers  
        .Where((c)=>c.City=="Redmond")  
        .Where((c)=>c.Credit>10000)  
        .OrderByDescending((c)=>c.Age)  
    select c.Name;
```

# Query expression compilation

```
var names =  
    from c in Customers  
        .Where((c)=>c.City=="Redmond")  
        .Where((c)=>c.Credit>10000)  
        .OrderByDescending((c)=>c.Age)  
    select c.Name;
```

# Query expression compilation

```
var names =  
    from c in Customers  
        .Where((c)=>c.City=="Redmond")  
        .Where((c)=>c.Credit>10000)  
        .OrderByDescending((c)=>c.Age)  
    select c.Name;
```

```
var names =  
    Customers  
        .Where((c)=>c.City=="Redmond")  
        .Where((c)=>c.Credit>10000)  
        .OrderByDescending((c)=>c.Age)  
        .Select((c)=>c.Name);
```

# The query expression pattern (subset)

```
class C<T>
{
    public C<T> Where(Func<T,bool> predicate);
    public C<U> Select<U>(Func<T,U> selector);
    public C<V> SelectMany<U,V>(Func<T,C<U>> selector,
        Func<T,C<U>,V> resultSelector);
    public C<V> Join<U,K,V>(C<U> inner, Func<T,K> outerKeySelector,
        Func<U,K> innerKeySelector, Func<T,U,V> resultSelector);
    public C<V> GroupJoin<U,K,V>(C<U> inner,
        Func<T,K> outerKeySelector,
        Func<U,K> innerKeySelector,
        Func<T,C<U>,V> resultSelector);
    public O<T> OrderBy<K>(Func<T,K> keySelector);
    public O<T> OrderByDescending<K>(Func<T,K> keySelector);
    public C<G<K,T>> GroupBy<K>(Func<T,K> keySelector);
    public C<G<K,E>> GroupBy<K,E>(Func<T,K> keySelector,
        Func<T,E> elementSelector);
}
```

# A step back

- Before we think about translating C# 3.0
- ... Let's think about C# 2.0 for now

# A step back

- Before we think about translating C#3.0
- ... Let's think about C# 2.0 for now
- .....but let's think about it formally!



# Typing relations in C<sup>#</sup>2.0

Checking	$\Gamma \vdash e \downarrow_i \tau$	“ $e$ can be implicitly converted to $\tau$ ”
	$\Gamma \vdash e \downarrow_x \tau$	“ $e$ can be explicitly converted to $\tau$ ”
	$\Gamma \vdash s \downarrow \phi$	“statement $s$ can be converted to $\phi$ ”
Synthesis	$\Gamma \vdash e \uparrow^s \phi$	“ $e$ synthesizes a type $\phi$ ”
Inference	$\Gamma; \bar{X} \vdash e \sim \tau \hookrightarrow \theta$	“ $e$ matches type $\tau$ with free type parameters $\bar{X}$ and infers the substitution $\theta$ ”

# Typing relations in C<sup>#</sup>2.0

Checking	$\Gamma \vdash e \downarrow_i \tau$	“ $e$ can be implicitly converted to $\tau$ ”
	$\Gamma \vdash e \downarrow_x \tau$	“ $e$ can be explicitly converted to $\tau$ ”
	$\Gamma \vdash s \downarrow \phi$	“statement $s$ can be converted to $\phi$ ”
Synthesis	$\Gamma \vdash e \uparrow^s \phi$	“ $e$ synthesizes a type $\phi$ ”
Inference	$\Gamma; \bar{X} \vdash e \sim \tau \leftrightarrow \theta$	“ $e$ matches type $\tau$ with free type parameters $\bar{X}$ and infers the substitution $\theta$ ”

## Fact

$\Gamma \vdash e \uparrow^s \tau$  iff  $\Gamma; X \vdash e \sim X \leftrightarrow [X \mapsto \tau]$

# Type synthesis relation

$$\boxed{\Gamma \vdash e \uparrow^s \tau}$$

$$\frac{}{\Gamma \vdash b \uparrow^s \text{bool}} \quad \frac{}{\Gamma \vdash i \uparrow^s \text{int}} \quad \frac{}{\Gamma, x: \tau \vdash x \uparrow^s \tau}$$
$$\frac{\Gamma \vdash e \downarrow_x \tau}{\Gamma \vdash (\tau)e \uparrow^s \tau} \quad \frac{\Gamma \vdash e \uparrow^s \tau_1 \quad \text{ftype}(\tau_1, f) = \tau_2}{\Gamma \vdash e.f \uparrow^s \tau_2}$$

# Type synthesis relation

$$\boxed{\Gamma \vdash e \uparrow^s \tau}$$

$$\frac{}{\Gamma \vdash b \uparrow^s \text{bool}} \quad \frac{}{\Gamma \vdash i \uparrow^s \text{int}} \quad \frac{}{\Gamma, x: \tau \vdash x \uparrow^s \tau}$$
$$\frac{\Gamma \vdash e \downarrow_x \tau}{\Gamma \vdash (\tau)e \uparrow^s \tau} \quad \frac{\Gamma \vdash e \uparrow^s \tau_1 \quad \text{ftype}(\tau_1, f) = \tau_2}{\Gamma \vdash e.f \uparrow^s \tau_2}$$

## Note

There are no synthesis rules for `null` or AMEs!

# Implicit conversion relation

$$\boxed{\Gamma \vdash e_1 \downarrow_i \tau_1}$$

$$\frac{}{\Gamma \vdash \text{null} \downarrow_i \rho} \quad \frac{d\text{type}(D)(\bar{\tau}) = \bar{\tau}_0 \rightarrow \phi \quad \Gamma, \bar{x}: \bar{\tau}_0 \vdash \bar{s} \downarrow \phi}{\Gamma \vdash \text{delegate}(\bar{\tau}_0 \bar{x}) \{\bar{s}\} \downarrow_i D\langle\bar{\tau}\rangle}$$

$$\frac{\Gamma \vdash e \uparrow^s \tau_1 \quad \tau_1 <: \tau_2}{\Gamma \vdash e \downarrow_i \tau_2}$$

$$\boxed{\Gamma \vdash e_1 \downarrow_x \tau_1}$$

$$\frac{\Gamma \vdash e \downarrow_i \tau}{\Gamma \vdash e \downarrow_x \tau} \quad \frac{\Gamma \vdash e \uparrow^s \tau_1 \quad \tau_1 <:^x \tau_2}{\Gamma \vdash e \downarrow_x \tau_2}$$

And now on to the translation!

# Typing relations (recall)

Checking  $\Gamma \vdash e \downarrow_i \tau$  “ $e$  can be implicitly converted to  $\tau$ ”

$\Gamma \vdash e \downarrow_x \tau$  “ $e$  can be explicitly converted to  $\tau$ ”

$\Gamma \vdash s \downarrow \phi$  “statement  $s$  can be converted to  $\phi$ ”

Synthesis  $\Gamma \vdash e \uparrow^s \phi$  “ $e$  synthesizes a type  $\phi$ ”

# Typing relations into translations

- Checking  $\Gamma \vdash e \downarrow_i \tau \rightsquigarrow e'$  “ $e$  can be implicitly converted to  $\tau$  yielding  $e'$ ”
- $\Gamma \vdash e \downarrow_x \tau \rightsquigarrow e'$  “ $e$  can be explicitly converted to  $\tau$  yielding  $e'$ ”
- $\Gamma \vdash s \downarrow \phi \rightsquigarrow s'$  “statement  $s$  can be converted to  $\phi$  yielding  $s'$ ”
- Synthesis  $\Gamma \vdash e \uparrow^s \phi \rightsquigarrow e'$  “ $e$  synthesizes a type  $\phi$  yielding  $e'$ ”

Key fact

**It's all about type-directed translation!**

# Simple example

$$\frac{dtype(D)(\bar{\tau}) = \bar{\tau}_0 \rightarrow \tau_1 \quad \Gamma, \bar{x}_0 : \bar{\tau}_0 \vdash e_1 \downarrow_i \tau_1 \rightsquigarrow e_{11}}{\Gamma \vdash (\bar{x}_0) \Rightarrow e_1 \downarrow_i D\langle\bar{\tau}\rangle \rightsquigarrow \text{delegate}(\bar{\tau}_0 \bar{x}) \{ \text{return } e_{11}; \}}$$

$$\frac{dtype(D)(\bar{\tau}) = \bar{\tau}_0 \rightarrow \tau_1 \quad \Gamma, \bar{x}_0: \bar{\tau}_0 \vdash e_1 \downarrow_i \tau_1 \rightsquigarrow e_{11}}{\Gamma \vdash (\bar{x}_0) \Rightarrow e_1 \downarrow_i D\langle\bar{\tau}\rangle \rightsquigarrow \text{delegate}(\bar{\tau}_0 \bar{x}) \{ \text{return } e_{11}; \}}$$

## Content

Lambda expressions are translated to AMEs

# Simple example

$$\frac{\Gamma \vdash e_1 \uparrow^s \tau_2 [ ] \rightsquigarrow e_{11} \quad x \notin \text{dom}(\Gamma) \quad \Gamma, x: \tau_2 \vdash s_1 \downarrow \phi \rightsquigarrow s_{11}}{\Gamma \vdash \text{foreach} (\text{var } x_1 \text{ in } e_1) s_1 \downarrow \phi \rightsquigarrow \text{foreach} (\tau_2 x_1 \text{ in } e_{11}) s_{11}}$$

# Synthesis/Conversion difference exposed

$$\frac{\Gamma \vdash e_1 \uparrow^s \tau_1 \rightsquigarrow e_{11} \quad x \notin \text{dom}(\Gamma) \quad \Gamma, x: \tau_1 \vdash \bar{s}_1 \downarrow \phi \rightsquigarrow \bar{s}_{11}}{\Gamma \vdash \text{var } x = e_1; \bar{s}_1 \downarrow \phi \rightsquigarrow \tau_1 x = e_{11}; \bar{s}_{11}}$$

$$\frac{\Gamma \vdash e_1 \downarrow_i \tau_1 \rightsquigarrow e_{11} \quad x \notin \text{dom}(\Gamma) \quad \Gamma, x: \tau_1 \vdash \bar{s}_1 \downarrow \phi \rightsquigarrow \bar{s}_{11}}{\Gamma \vdash \tau_1 x = e_1; \bar{s}_1 \downarrow \phi \rightsquigarrow \tau_1 x = e_{11}; \bar{s}_{11}}$$

# Synthesis/Conversion difference exposed

$$\frac{\Gamma \vdash e_1 \uparrow^s \tau_1 \rightsquigarrow e_{11} \quad x \notin \text{dom}(\Gamma) \quad \Gamma, x: \tau_1 \vdash \bar{s}_1 \downarrow \phi \rightsquigarrow \bar{s}_{11}}{\Gamma \vdash \text{var } x = e_1; \bar{s}_1 \downarrow \phi \rightsquigarrow \tau_1 x = e_{11}; \bar{s}_{11}}$$

$$\frac{\Gamma \vdash e_1 \downarrow_i \tau_1 \rightsquigarrow e_{11} \quad x \notin \text{dom}(\Gamma) \quad \Gamma, x: \tau_1 \vdash \bar{s}_1 \downarrow \phi \rightsquigarrow \bar{s}_{11}}{\Gamma \vdash \tau_1 x = e_1; \bar{s}_1 \downarrow \phi \rightsquigarrow \tau_1 x = e_{11}; \bar{s}_{11}}$$

All these fail!

```
var a = null;  
var b =  
  delegate(int x){return x;};  
var c = (x)=> x;
```

# Synthesis/Conversion difference exposed

$$\frac{\Gamma \vdash e_1 \uparrow^s \tau_1 \rightsquigarrow e_{11} \quad x \notin \text{dom}(\Gamma) \quad \Gamma, x: \tau_1 \vdash \bar{s}_1 \downarrow \phi \rightsquigarrow \bar{s}_{11}}{\Gamma \vdash \text{var } x = e_1; \bar{s}_1 \downarrow \phi \rightsquigarrow \tau_1 x = e_{11}; \bar{s}_{11}}$$

$$\frac{\Gamma \vdash e_1 \downarrow_i \tau_1 \rightsquigarrow e_{11} \quad x \notin \text{dom}(\Gamma) \quad \Gamma, x: \tau_1 \vdash \bar{s}_1 \downarrow \phi \rightsquigarrow \bar{s}_{11}}{\Gamma \vdash \tau_1 x = e_1; \bar{s}_1 \downarrow \phi \rightsquigarrow \tau_1 x = e_{11}; \bar{s}_{11}}$$

## All these fail!

```
var a = null;  
var b =  
    delegate(int x){return x;};  
var c = (x)=> x;
```

## All these work!

```
Button a = null;  
Func<int,int> b =  
    delegate(int x){return x;};  
Func<Button,object> c = (x)=> x;
```

# An awkward wart

There's a nasty with type inference and overloading resolution:

```
static void mybar<T>(int a, T b){ Console.WriteLine("1"); }  
static void mybar<T>(int a, int b){ Console.WriteLine("2"); }
```

# An awkward wart

There's a nasty with type inference and overloading resolution:

```
static void mybar<T>(int a, T b){ Console.WriteLine("1"); }  
static void mybar<T>(int a, int b){ Console.WriteLine("2"); }
```

```
mybar(42,42);
```

# An awkward wart

There's a nasty with type inference and overloading resolution:

```
static void mybar<T>(int a, T b){ Console.WriteLine("1"); }  
static void mybar<T>(int a, int b){ Console.WriteLine("2"); }
```

```
mybar(42,42);           // Infers <int>; Prints 1
```

# An awkward wart

There's a nasty with type inference and overloading resolution:

```
static void mybar<T>(int a, T b){ Console.WriteLine("1"); }  
static void mybar<T>(int a, int b){ Console.WriteLine("2"); }
```

```
mybar(42,42);           // Infers <int>; Prints 1
```

```
mybar<int>(42,42);
```

# An awkward wart

There's a nasty with type inference and overloading resolution:

```
static void mybar<T>(int a, T b){ Console.WriteLine("1"); }  
static void mybar<T>(int a, int b){ Console.WriteLine("2"); }
```

```
mybar(42,42);           // Infers <int>; Prints 1
```

```
mybar<int>(42,42);     // Prints 2 :-(  

```

Normally we'd formalize local type inference as, e.g.

$$\Gamma \vdash e.m(\bar{f}) : \sigma \rightsquigarrow e.m\langle\bar{\tau}\rangle(\bar{f})$$

**But this doesn't work** ☹

Several solutions possible; we use method descriptors from MSIL

# Conclusions

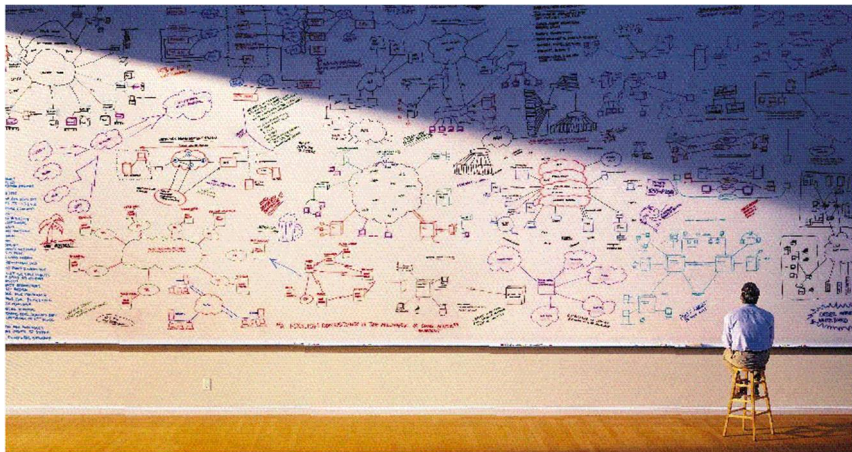
- The LINQ project enables query-like facilities into the .NET languages
- C# 3.0 (and VB 9.0) contains a number of FP-inspired extensions to enhance LINQ programming
- These new language features can be explained by type-directed translations (this paper)
- Formal techniques are useful specification tools 😊
  - Machine-assisted tools would be great here!

Far too much to mention!

- Close relative:  $C\omega$  [Bierman, Meijer, Schulte]
- Lots and lots of other work (much academic):

SQLJ, XJ, TL, HaskellDB, Fibonacci, Galileo, XDuce, CDuce, Napier88, OPJ, PJama, Thor, . . .

# Questions?



Or come to the Microsoft booth!