

# Compiler plugins for LINQ

Miguel Garcia



<http://www.sts.tu-harburg.de/people/mi.garcia>



# Agenda

- The long road to DPBLs
- LINQ: what's new and what's not
- Denotational semantics, “the missing link in LINQ”
- A tale of two typesafe rewritings:
  - LINQ nested in Java
  - LINQ nested in Scala
- Future work: Scala (with LINQ) as a DBPL

# The long road to DPBLs

- Having the best DBMS and the best programming language is not enough
- We need in fact:
  - a persistence-aware compiler, and
  - a programming-language aware DBMS
- DBPL = control flow +  
DDL + query language + DML
- MS decided a new query language was needed  
**LINQ**

# LINQ: A query language with ...

- A functional-object flavor
- Few “built-in” constructs (join, group by, orderby)
- Most functionality provided by
  - Object-based functions
  - In particular, those on collections: sum, min, max, etc.
- Lean design:
  - Comprehensions and closures
  - Shorthand syntax for grouping and sorting
  - And that’s all, really

## Warm-up examples (C# syntax where it makes a difference)

LINQ	<pre>from x1 in e1 from x2 in e2 where cond select e3</pre>
Math	<pre>[ e3   x1 ← e1, x2 ← e2, cond ]</pre>
Scala	<pre>for (x1 &lt;- e1; x2 &lt;- e2; if cond) yield (e3)</pre>

## Warm-up examples (C# syntax where it makes a difference)

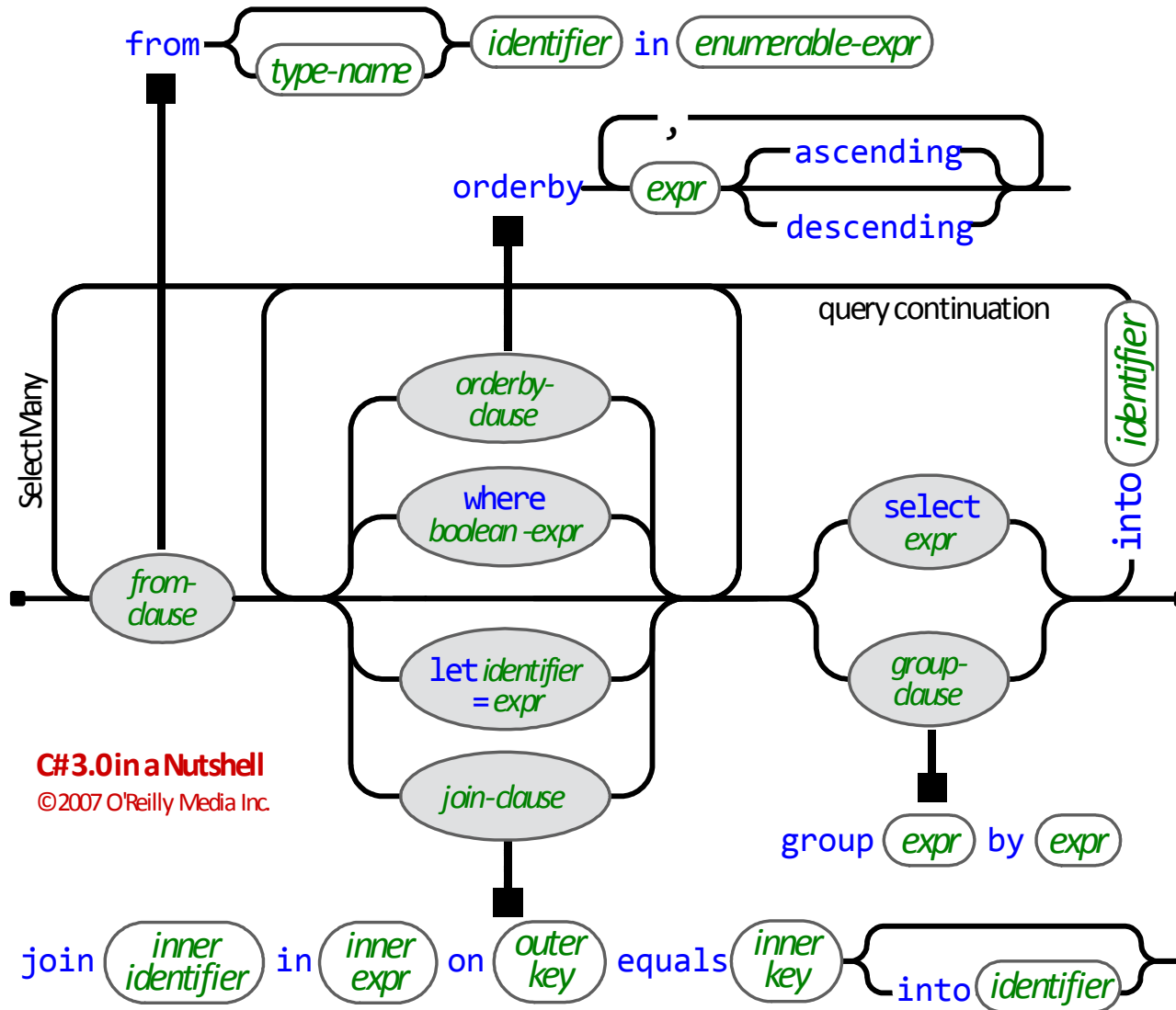
<b>LINQ</b>	<pre>from x1 in e1 from x2 in e2 where cond select e3</pre>
<b>Math</b>	<pre>[ e3   x1 ← e1, x2 ← e2, cond ]</pre>
<b>Scala</b>	<pre>for (x1 &lt;- e1; x2 &lt;- e2; if cond) yield (e3)</pre>

<b>LINQ</b>	<pre>from c in customers group c.name by c.country;</pre>
<b>Math</b>	<pre>[ (country, [c.name   c ← customers,               c.country == country] )     country ← noDup([c.country c ← customers])]</pre>
<b>Scala</b>	<pre>val countries = noDup(customers map (_.country)) for (country &lt;- countries) yield (country,   customers filter (_.country == country) map (_.name) )</pre>

# What we're focusing on

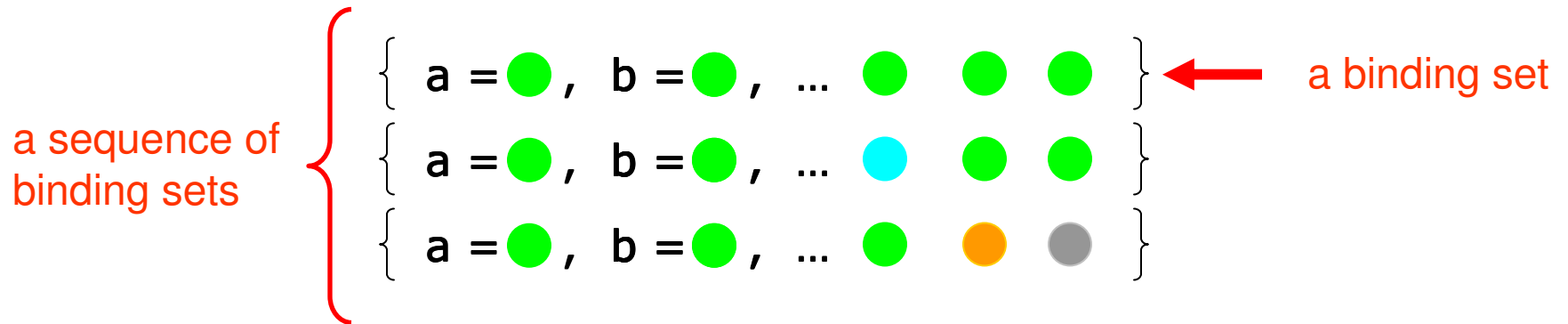
- Wouldn't it be great if LINQ snippets could appear nested in Java or Scala code?
  - That's what this talk is about
  - We'll see how LINQ is desugared into functional-style building blocks (the “SQO” operators, or if you will, the more fundamental *flat*, *filter*, and *flatMap* operators)
- What this talk does not cover:
  - Technologies related to LINQ, in particular the Entity Framework (responsible for most of what we perceive as “LINQ power”)
  - Query optimization
  - Communication with a DBMS
- But first of all, some good old denotational semantics

# Main points about LINQ syntax



Railroad diagram for the textual syntax of LINQ, reproduced from <http://www.albahari.com/nutshell/linqsyntax.html>

# Denotational semantics in 5 minutes



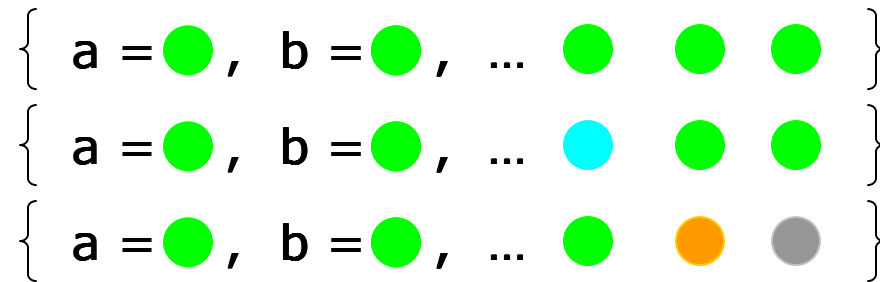
↓ fed into a *Select* valuation function

$$\llbracket \text{select } E_{selexp} \rrbracket envs \stackrel{\text{def}}{=} [ \llbracket selexp \rrbracket (env) \mid env \leftarrow envs ]$$

↓ results in

a result sequence  
(no more binding sets)

# Other valuations just enrich an environment



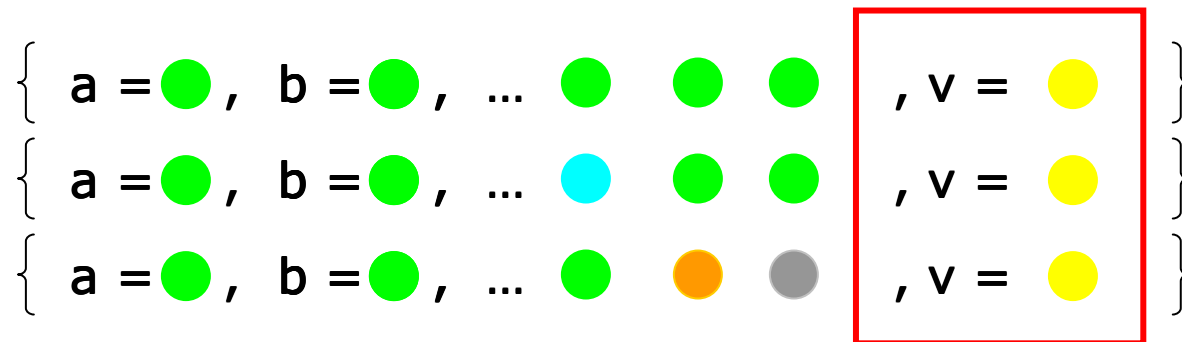
fed into a *Let* valuation function

$$\llbracket \text{let } V_{var} = E_{exp} \rrbracket envs \stackrel{\text{def}}{=} [env' \mid env \leftarrow envs, \\ \text{let } env' = env \cup \{var \mapsto \llbracket exp \rrbracket (env)\} ]$$



results in

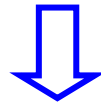
**another**  
sequence of  
binding sets,  
for consumption  
by the next  
LINQ construct



# Desugaring LINQ into SQO: the big picture

## “LINQ textual syntax”

```
var q = from p in people where p.Age > 20
        orderby p.Age descending, p.Name
        group new {
            p.Name, Senior = p.Age > 30, FirstJob = p.Jobs[0]
        } by p.CanCode;
```



```
var q = people.Where(p => p.Age > 20)
               .OrderByDescending(p => p.Age).ThenBy(p => p.Name)
               .GroupBy(p => p.CanCode,
                        p => new { p.Name, Senior = p.Age > 30,
                                   FirstJob = p.Jobs[0]
                        } );
```

## “Standard Query Operators” (SQO)

# SQO: 50 operators with 94 overloaded signatures (4 shown below)

<pre>public static IQueryable&lt;IGrouping&lt;TKey, TSource&gt;&gt; GroupBy&lt;TSource, TKey&gt;(     this IQueryable&lt;TSource&gt; source,     Expression&lt;Func&lt;TSource, TKey&gt;&gt;     keySelector)</pre>	Groups the elements of a sequence according to a specified key selector function.
<pre>public static IQueryable&lt;IGrouping&lt;TKey, TSource&gt;&gt; GroupBy&lt;TSource, TKey&gt;(     this IQueryable&lt;TSource&gt; source,     Expression&lt;Func&lt;TSource, TKey&gt;&gt; keySelector,     IEqualityComparer&lt;TKey&gt; comparer)</pre>	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
<pre>public static IQueryable&lt;IGrouping&lt;TKey, TElement&gt;&gt; GroupBy&lt;TSource, TKey, TElement&gt;(     this IQueryable&lt;TSource&gt; source,     Expression&lt;Func&lt;TSource, TKey&gt;&gt;     keySelector,     Expression&lt;Func&lt;TSource, TElement&gt;&gt;     elementSelector )</pre>	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
<pre>public static IQueryable&lt;TResult&gt; GroupBy&lt;TSource, TKey, TResult&gt;(     this IQueryable&lt;TSource&gt; source,     Expression&lt;Func&lt;TSource, TKey&gt;&gt;     keySelector,     Expression&lt;Func&lt;TKey, IEnumerable&lt;TSource&gt;,     TResult&gt;&gt; resultSelector )</pre>	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.

How to pick the correct ones when desugaring from LINQ?

# 18 transformation rules to the rescue

ID	Phase	Description	§ in C# spec.	
T1	1	Inline query continuation	7.15.2.1	
T2	2	<i>FromClause</i>	7.15.2.2	
T3		<i>JoinClause</i>		
T4		<i>JoinIntoClause</i>		
T5	3	Identity query	7.15.2.3	
T6	4	<i>FromClause FromClause</i>	<i>SelectClause</i>	
T7			otherwise	
T8		<i>FromClause JoinClause</i>	<i>SelectClause</i>	
T9			otherwise	
T10		<i>FromClause JoinIntoClause</i>	<i>SelectClause</i>	7.15.2.4
T11			otherwise	
T12			<i>OrderByClause</i>	7.15.2.5
T13			<i>WhereClause</i>	
T14		<i>LetClause</i>		
T15		<i>FromClause</i>	non-identity <i>SelectClause</i>	
T16			identity <i>SelectClause</i>	
T17			non-identity <i>GroupByClause</i>	
T18			identity <i>GroupByClause</i>	

(details in the paper)

Implemented in LINQExpand4Java,  
an open-source JSR-269 compiler plugin

# T1: Inline query continuations

(a)	T1: inline query continuation from x1 in e1 ... into x2 ... → from x2 in (from x1 in e1 ...) ...
(b)	function T1 ( q : QueryExp ) : QueryExp when q.qbody.qcont ≠ ∅
(c)	new QueryExp { from = new FromClause { type = q.qbody.qcont.type, var = q.qbody.qcont.var, in = new QueryExp { from = q.from qbody = new QueryBody { qbclauses = q.qbody.qbclauses, sel_gby = q.qbody.sel_gby } } } qbody = q.qbody.qcont.qbody }

## How does Scala compare to SQO as translation target?

- Pros: Avoids learning the 90+ overloaded SQO operators
- Cons: Grouping and sorting (where Scala lacks syntax shorthands) result in less-than-compact expressions

T10	From followed by a join with into followed by a select
LINQ	<pre>from x1 in e1 join x2 in e2 on k1 equals k2 into g select e3</pre>
SQO	<pre>e1.GroupJoin( e2,               x1 =&gt; k1, x2 =&gt; k2,               (x1, g) =&gt; e3 )</pre>
Scala	<pre>for ( x1 &lt;- e1; val outerKey = k1;       val g = e2 filter { x2 =&gt; outerKey == k2 }     ) yield e3</pre>

# More pros

- The Scala compiler performs its own desugaring
- Additional processing (e.g., optimization) possible via flexible compiler plugin architecture (better than Java's)

Scala	<pre>val res = for (x1 &lt;- e1; x2 &lt;- e2) yield (x1 + x2)</pre>
Desugar	<pre>val res: Array[Double] =   e1.flatMap[Double](((x1: Double) =&gt;     e2.map[Double](((x2: Double) =&gt; x1.+(x2)))));</pre>

- AST rewriting can rely on the Kiama library
  - <http://code.google.com/p/kiama>
- Allows expressing in Scala two program-transformation paradigms (in a manner faithful to their original notations):
  - strategic term rewriting, i.e. Stratego
  - (circular) attribute grammars, i.e. JastAdd

## ... and a few (transient) cons too

- As of Scala 2.7.5, no partial evaluation performed by default
  - Either implement it yourself, or
  - Try compiling with `-optimise`, `-specialize`, `-experimental`, or
  - Wait for Scala 2.8

<b>Scala</b>	<pre>val res2 = Set(1, 2, 3) map (x =&gt; x)</pre>
<b>Desugar</b>	<p>Not reduced to « Set(1, 2, 3) » The identity function will be applied at runtime, somewhere in the snippet below J</p> <pre>val res2: scala.collection.immutable.Set = scala.this.Predef.Set().apply( new scala.runtime.BoxedIntArray( Array[Int]{1,2,3}) ) .map( {(new QueryDesugar\$\$anonfun\$2(): Function1)})</pre>

# Going the last mile to make Scala a DBPL

- Our first prototype:
  - Typesafe rewriting of nested LINQ into (main-memory only) Scala
- Further down the road
  - Deciding on a rich subset of Scala comprehensions for translation into a DB query language
  - Using a comprehensions-aware query planner:
    - lambda-DB, <http://lambda.uta.edu/lambda-DB/manual/>
    - Ferry & Pathfinder, <http://www-db.informatik.uni-tuebingen.de/research/ferry>
    - Links, <http://www.inf.ed.ac.uk/publications/report/1327.html>
- And the laundry list goes on
  - DML, caching, events, materialized views, parallelism

# Conclusions

- A “proof of concept” DPBL can be grown one component at a time (working hypothesis)
  - Some sub-problems have great solutions done by others
- Be wary of any new query language lacking denotational semantics
- Scala is here to stay, also in the field of OODBs
  - At least, for as long as LINQ remains relevant in the field of OODBs