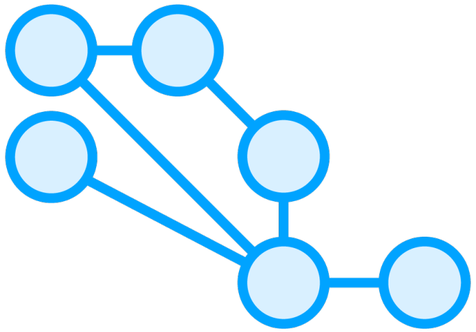
Understanding Expression Trees in C#

Chris B. Behrens

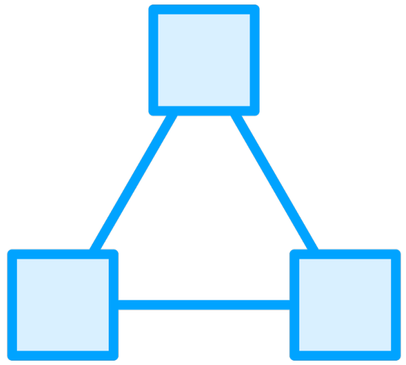
@chrisbbehrens

LINQ uses lambda expressions to build an expression tree, which it can then use to build a provider-specific query.

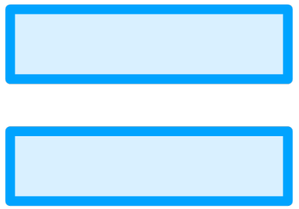
LINQ Providers



LINQ to x



The expression tree is independent of the implementation



Equivalent schemas have equivalent queries

Demo

A simple object collection

Write a simple lambda to filter and order it

Extract that lambda to a function

Talk about what that means

Two Ideas

An expression can mean entirely different things in different contexts

The lambda is entirely separable from its use

Introducing LINQPad

What LINQPad Is

A LINQ debug
and diagnostics tool

Provide the context and
execute LINQ against it

Demo

The LINQPad interface

Review a couple of sample queries and their output

We'll use our sample project as an execution context

Execute our program

Look at the results

Where We're Headed

That tree is built off
our code in the window

What if the user
could drive the tree?

Why Do We Need Expressions?

We could just code a branch for each selection

But this creates a huge number of combinations

We need a better way

IEnumerable

The System.LINQ Namespace

You begin writing a WHERE clause...

And then you realize that you need to include the namespace

What *is it* that implements IEnumerable?

```
var electricGuitars = guitars.Where(guitar => guitar.Pickup == PickupType.Electric);
```

```
IEnumerable<Guitar> electricGuitars = Enumerable.Where<Guitar>(guitars, guitar =>
guitar.Pickup == PickupType.Electric);
```

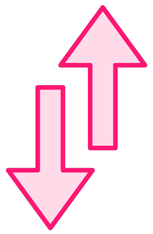
Demo

IEnumerable

Where it gets implemented

IQueryable

What This Means



Two interfaces that have the same methods, but do different things

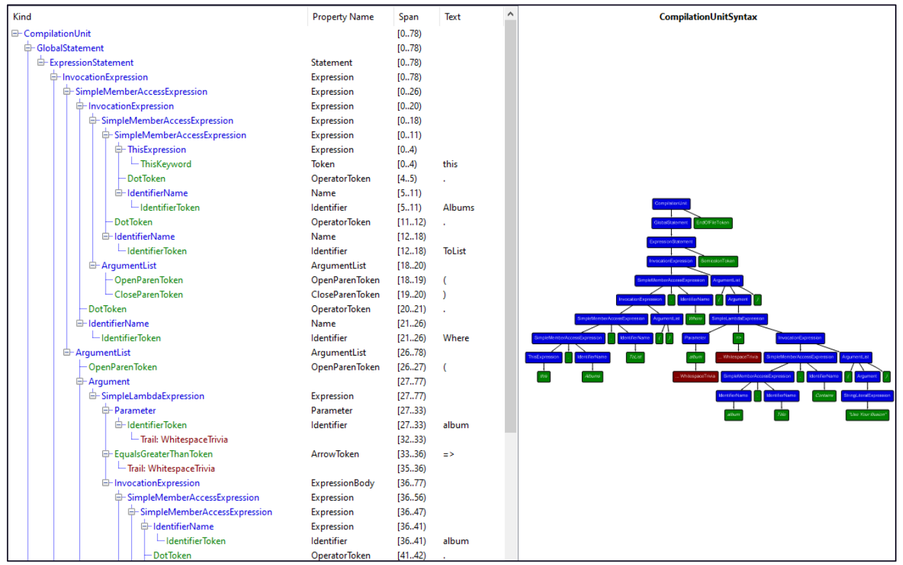
IEnumerable is for in-memory manipulation

IQueryable is for interacting with external systems

LINQ uses lambda expressions to build an expression tree, which it can then use to build a provider-specific query.

Types of Expressions

System.LINQ.Expression



Let’s get our hands dirty

Look at the shape of the tree on the left

Aside from the comments, whitespace, and angle brackets…

Most of those blocks you saw in the tree view are declarable types in .NET

The simplest way to construct the tree is to write the code…

But we’re going to construct it programmatically

Why Would You Do That?

Maybe you're just super-curious, like me

Like we said earlier, understanding how stuff works under the covers is important

Our ToList example proved that

Changing our runtime to someone else's design time

A dynamic filter

Expressions linked together with ANDs or ORs

A Little Deeper

We can make more sophisticated trees

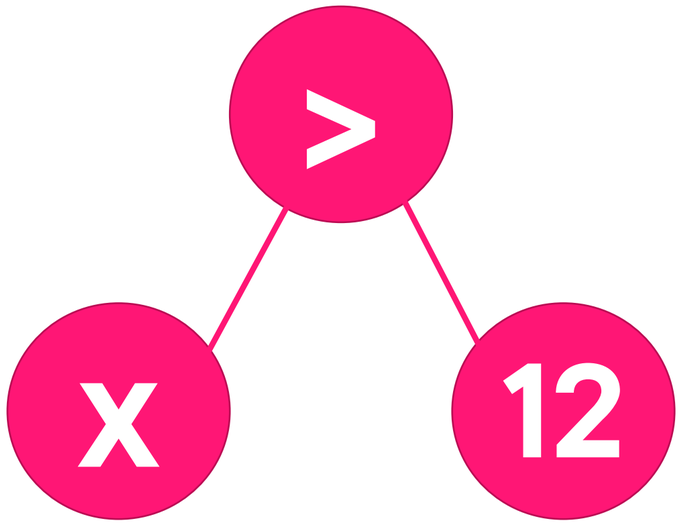
We could implement our own parser

And construct the tree from that

We could emit an entirely different language

The Expression Class

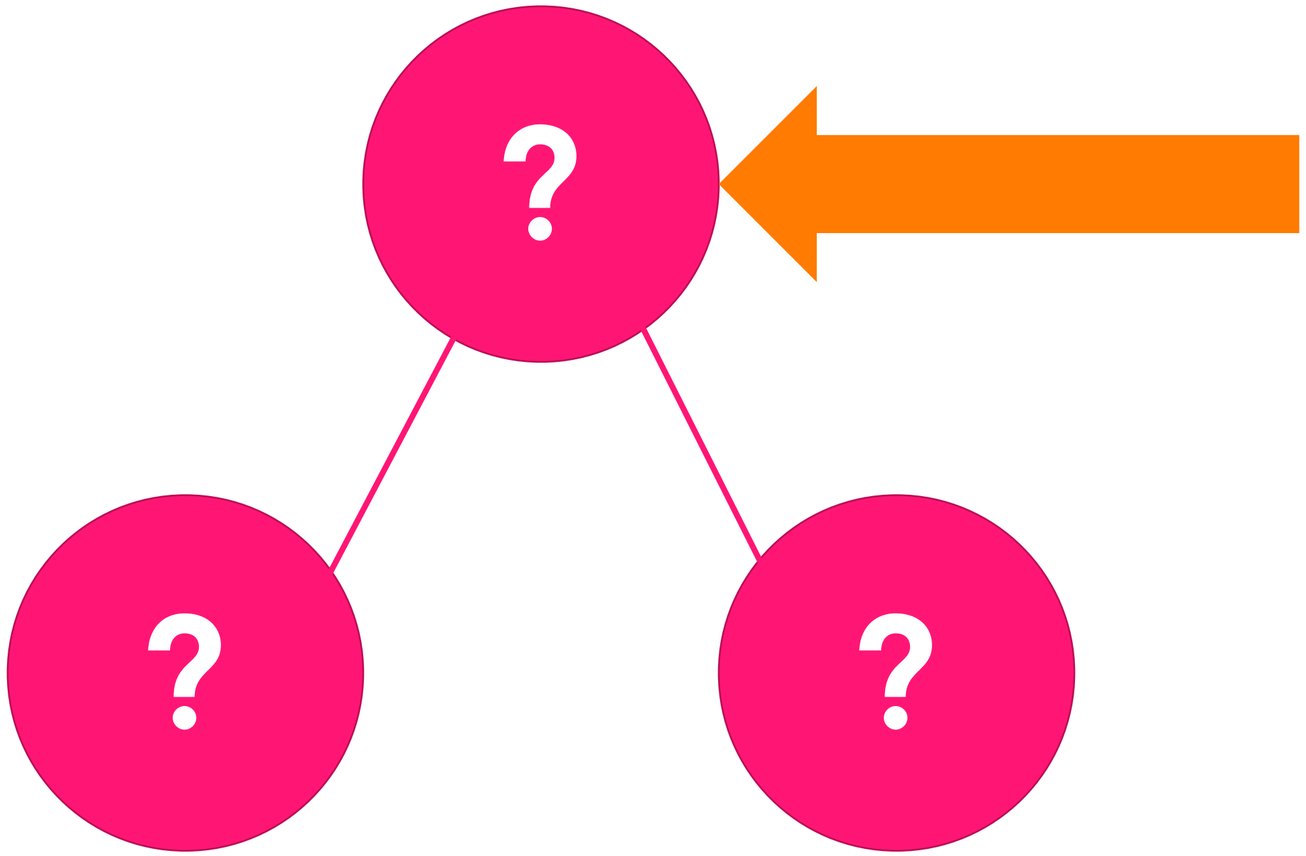
$$x > 12$$



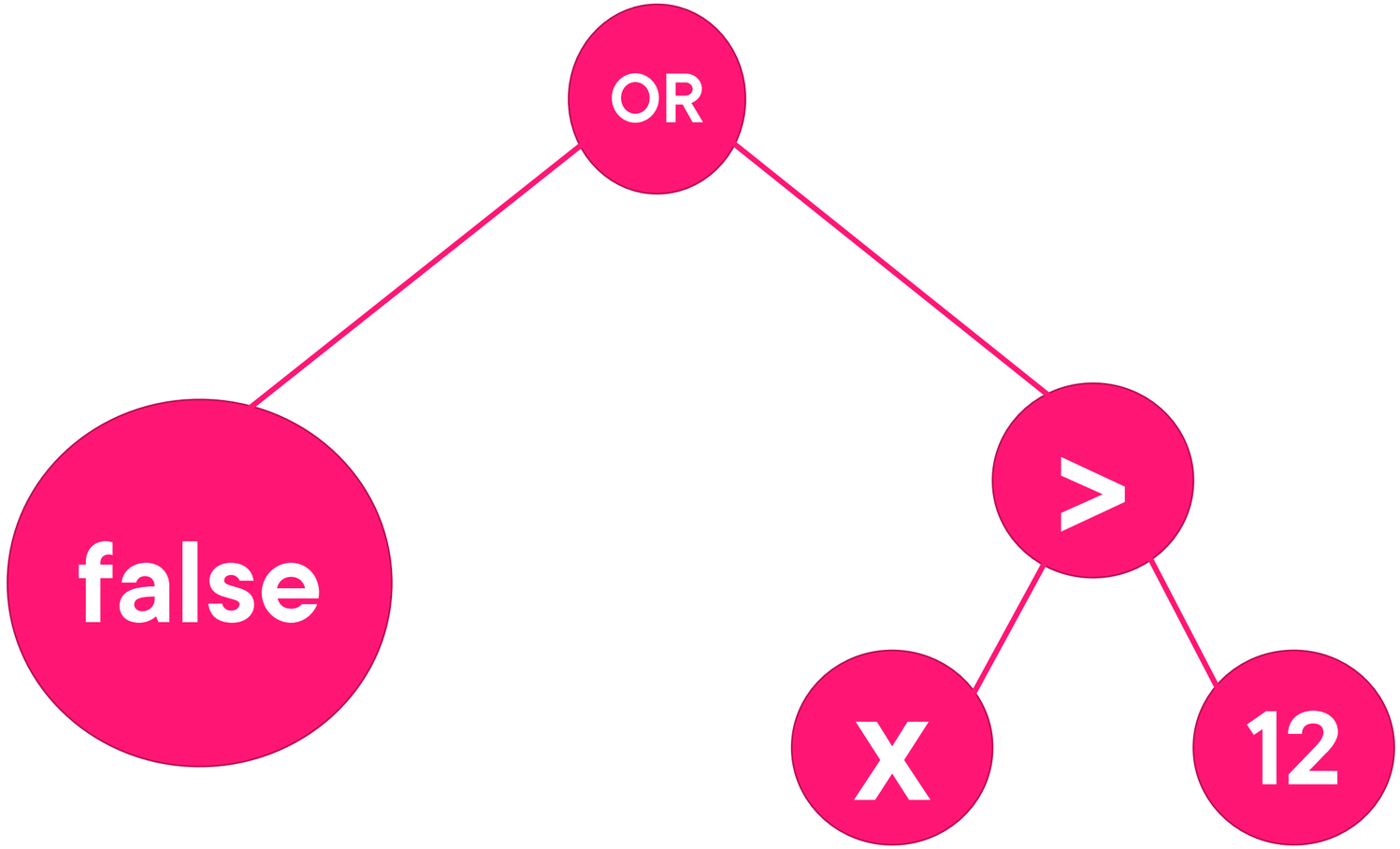
A Tree with More Branches

We can make our logic more complex by adding parents and peers to our tree

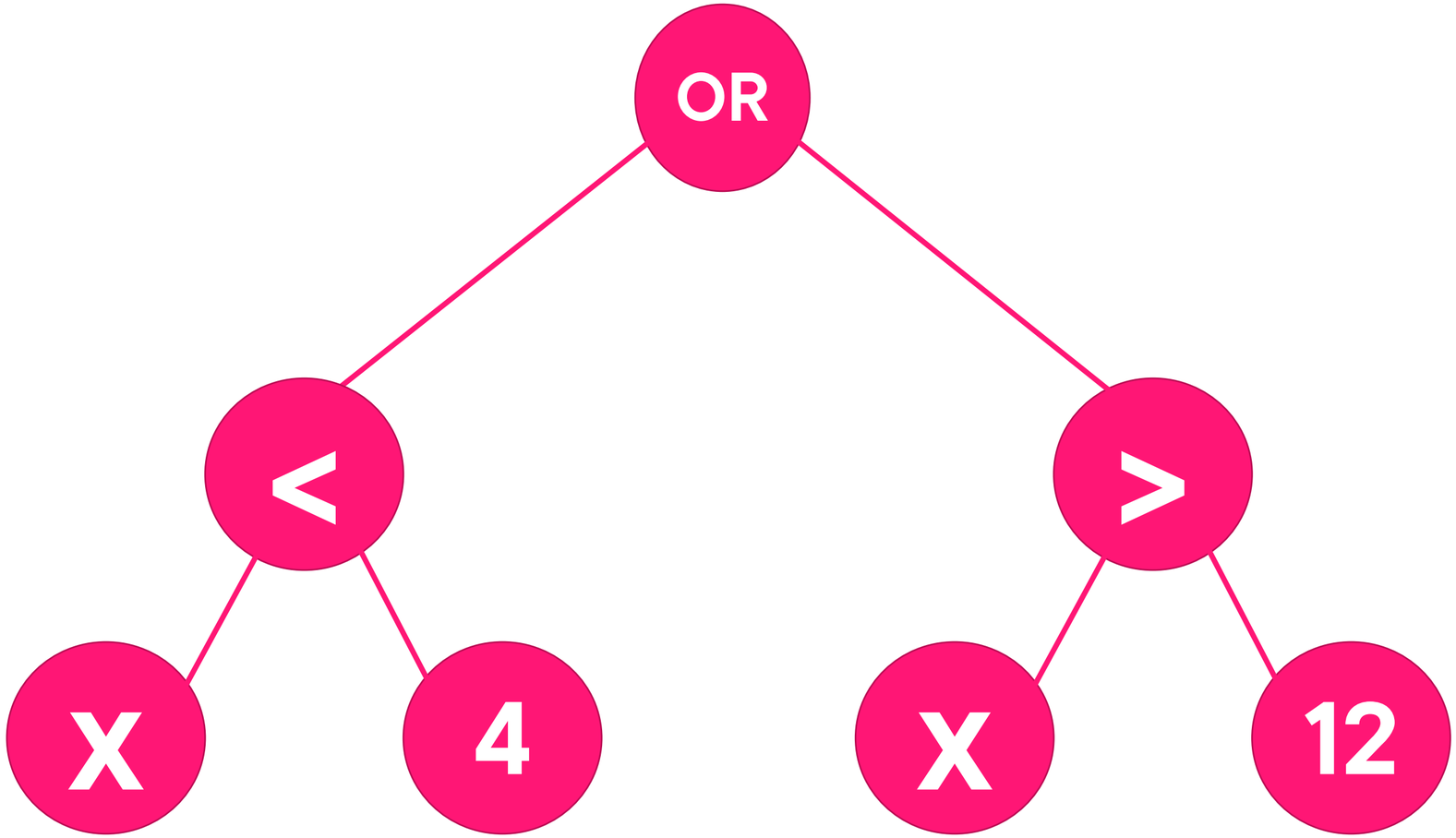
‘Operator’ Expressions



Adding Another Clause to Our Logic



Adding Another Clause to Our Logic



Summary

LINQ

How it works with lambda expressions

We extracted our lambda

LINQPad

IEnumerable and IQueryable

Expressions

Expressing a simple test of a value x as an expression tree