Empty Distribution

In this lesson, we will implement the empty distribution.

WE'LL COVER THE FOLLOWING

Previously, we mentioned that we will be needing the empty distribution.

Implementing the Empty Distribution

In this lesson, we will be implementing the Empty Distribution.

```
public sealed class Empty<T> : IDiscreteDistribution<T>
{
  public static readonly Empty<T> Distribution = new Empty<T>();
  private Empty() { }
  public T Sample() => throw new Exception("Cannot sample from empty distr
ibution");
  public IEnumerable<T> Support() => Enumerable.Empty<T>();
  public int Weight(T t) => 0;
}
```

Now that we have this, we can fix up our other distributions to use it. The `WeightedInteger` factory becomes:

```
public static IDiscreteDistribution<int> Distribution(IEnumerable<int> wei
ghts)
{
  List<int> w = weights.ToList();

  if (w.Any(x => x < 0))
    throw new ArgumentException();

  if (!w.Any(x => x > 0))
```

```
    return Empty<int>.Distribution;


[...]
```

And the `Bernoulli` factory becomes:

```
public static IDiscreteDistribution<int> Distribution(int zero, int one)
{
  if (zero < 0 || one < 0)
    throw new ArgumentException();

  if (zero == 0 && one == 0)
    return Empty<int>.Distribution;

[...]
```

And the `StandardDiscreteUniform` factory becomes:

```
public static IDiscreteDistribution<int> Distribution(int min, int max)
{
  if (min > max)
    return Empty<int>.Distribution;

[...]
```

And the `Projected` factory becomes:

```
public static IDiscreteDistribution<R> Distribution(IDiscreteDistribution<
A> underlying, Func<A, R> projection)
{
  var result = new Projected<A, R>(underlying, projection);

  if (result.weights.Count == 0)
    return Empty<R>.Distribution;

[...]
}
```

And one more thing needs to change. Our computation in `SelectMany` assumed that none of the total weights are zero. Easily fixed:

```
int lcm = prior.Support()
```

```
  .Select(a => likelihood(a).TotalWeight())
  .Where(x => x != 0)
  .LCM();
```

We also have a division by total weight; don't we have to worry about dividing by zero? Nope. Remember, the empty distribution's support is the empty sequence, so when we then say:

```
var w = from a in prior.Support()
        let pb = likelihood(a)
        from b in pb.Support()
        group prior.Weight(a) * pb.Weight(b) *
          lcm / pb.TotalWeight()
        by projection(a, b);
```

If `prior.Support()` is empty, then the whole query is empty and so the division is never executed. If `prior.Support()` is not empty but one of the `pb.Support()` is empty then there is nothing from which to compute a group key. We never actually divide by total weight, and so there is no division by zero error to avoid.

That was relatively painless, but it is probably still very unclear why we'd ever need an empty distribution. It seems to be like a little bomb hiding in the program, waiting for someone to sample it. Have we just committed another "null reference" design fault? In a few lessons, we'll see what benefits justify the costs.

Implementation

Here is the complete implementation:

> We have added the empty distribution to implementation of the lesson Discrete Probability Distribution Type as a Monad

The output remains the same, however, some parts of the code have changed.

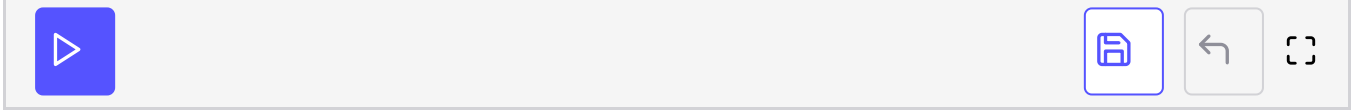
Program.cs

Bernoulli.cs

BetterRandom.cs

```
using System;
using System.Collections.Generic;

// Let's show how we can create Where out of SelectMany on sequences.
namespace Weird
{
    static class MyLinq
    {
        // Standard implementation of Select:
        public static IEnumerable<R> Select<A, R>(
            this IEnumerable<A> items,
            Func<A, R> projection)
        {
            foreach (A item in items)
                yield return projection(item);
        }

        // Standard implementation of SelectMany:
        public static IEnumerable<R> SelectMany<A, B, R>(
            this IEnumerable<A> items,
            Func<A, IEnumerable<B>> selection,
            Func<A, B, R> projection)
        {
            foreach (A a in items)
                foreach (B b in selection(a))
                    yield return projection(a, b);
        }

        public static IEnumerable<T> Single<T>(T t)
        {
            yield return t;
        }

        public static IEnumerable<T> Zero<T>()
```

```
        {
            yield break;
        }

        // Non-standard Where:
        public static IEnumerable<T> Where<T>(
                this IEnumerable<T> items,
                Func<T, bool> predicate) =>
            from a in items
            from b in predicate(a) ? Single(a) : Zero<T>()
            select b;
    }
}
namespace Probability
{
    // No using System.Linq.
    using Weird;
    static class Episode17
    {
        public static void DoIt()
        {
            Console.WriteLine("Episode 17");
            Console.WriteLine("Custom Where using only SelectMany");
            Console.WriteLine("aBcDe".Where(char.IsLower).CommaSeparated());
            Console.WriteLine("Delayed throw in enumerator block");
            var foo = Foo(null);
            Console.WriteLine("No throw yet!");
            try
            {
                foreach (int x in foo)
                    Console.WriteLine(x);
            }
            catch
            {
                Console.WriteLine("Now we throw.");
            }
        }

        static IEnumerable<int> Foo(string bar)
        {
            if (bar == null)
                throw new ArgumentNullException();
            yield return bar.Length;
        }
    }
}
```



We've been having a lot of fun treating distributions as monads that we can use query comprehensions on, but is that really the best possible syntax? We will consider this question over the next three lessons.