Which features should be supported in Lambda Project?

- We determine a type of lambda based on its context.
- P1, P2, … for task priorities
- D1, D2, … for comparing the difficulties (D1 is easier than D2)
- Consider following user code.

```
private class PrivateClass {
        ...
}

class Program {
        int a = 50;
        delegate int del();
        delegate V generic_del<T, V>(T x);
        Func<int> global_f = () => 100;

        static int funcfn(Func<int> f) { return f(); }

        static int delfn(del d) { return d(); }

        static T get_nth(T[] arr, int n) { return arr[n]; }

        static T generic_funcfn<T>(Func<T> f) {       return f(); }

        static int plus_a(int x) { return x + a; }

        static void Main(string[] args)
        {
                int[] arr = new int[]{1, 2, 4};
                // break point here
        }
}
```

- System.Func typed lambda without access to any variables and methods
- Example: (Func<int>)(() => 50)
- Example: funcfn(() => 50)
- Already supported

- Cast to user-defined delegate lambda
- Exmaple: (del)(() => 50)

- Currently, `Could not resolve type: Program.del` error happens
- P2 D1

● User-defined delegate lambda
- Example: delfn(() => 50)
- How#1: Add references to user code when compiling lambda, but it doesn't work if the delegate is private.
- How#2: Compile the lambda as System.Func, then do `new del(f)`
- P1 D2 / P1 D3 (private)
- Already supported (only public/not merged)

● Action/Predicate delegate lambda
- Example: (Action<int>)(x => System.Console.WriteLine(x))
- P1 D1
- Already supported

● Array that contains generic type as a method parameter
- Example: arr.IndexOf(arr, 4) // Array.IndexOf<T>(T[], T)
- Example: get_nth(arr, 0)
- Example: Array.Find(arr, x => x == 4) // Array.Find<T> (T[], Predicate<T>)
- P1? D1
  Currently, we don't support Array that contains generic type as a method parameter. This feature is not related to lambda project, but I think users would like to use `Array.Find` with lambda.

● Assignment
- Example: global_f = () => 200;
- P2 D1?
- it's possible to realize this, but it would not work as expected yet.

```
a = 5;
f = () => a == 5;
a = 40;
f.Invoke()   ----- X
```

  X must be `false` when we execute expressions above in C#, but X would be `true` if I implement assignment left-hand-side lambdas straightforwardly.

● Lambda that accesses to local/member/static-class variables <u>without side effect</u>
- Example: funcfn(() => a + 100) // 150
- P1 D2 / P1 D3 (if roslyn involved)
- Already supported

- Lambda that accesses to public method/class
- Example: funcfn(() => plus_a(50)) // 100
- P1 D2 / P1 D3 (same/similar as above)
- Already supported

- Lambda that accesses to private class
- Example: ...
- P3 D4 (how?)
- I have not implemented yet, but we have a possible solution
    1. [when compiling]
       Skip roslyn's visibility check
        - https://github.com/dotnet/roslyn/issues/15557#issuecomment-263560079
    2. [when invoking]
       Create new SDWP to invoke lambda with skipping visibility
        - https://github.com/mono/mono/blob/master/mono/metadata/class-internals.h#L87

- Lambda which contains a generic parameter
- Example: generic_funcfn(() => "example")
- Example: generic_funcfn(() => 0.55)
- Example: generic_del(x => typeof(x))
- Example: generic_del(x => x.ToString())
  Note that methods like
  `static T generic_funcfn2 <T>(T x, Func<T, T> f) { return f(x); }` are supported since the generic argument `T` depend on the first argument `x`. But methods like `generic_funcfn` are hard to support since we have
  to infer argument types and return type of the lambda.
- P3 D3 (user could explicitly give their types)

- Side effect
- Example: funcfn(() => { a += 50; return ++a; }) // 'a' must be 101 after this
- P4 D4

- Use DynamicMethod instead of Assembly.Load
- P3 D1

- Aync lambda