Result Builder Methods

To be useful as a result builder, a result builder type must provide a sufficient subset of function-building methods that enable the transformation of various statement kinds (`if`, `switch`, `for`..`in`, etc.). The following example result builder illustrates the various function-building methods one can define:

```
@resultBuilder
struct ExampleResultBuilder {
  /// The type of individual statement expressions in the transformed function,
  /// which defaults to Component if buildExpression() is not provided.
  typealias Expression = ...

  /// The type of a partial result, which will be carried through all of the
  /// build functions.
  typealias Component = ...

  /// The type of the final returned result, which defaults to Component if
  /// buildFinalResult() is not provided.
  typealias Result = ...

  /// Required by every result builder to build combined results from
  /// statement blocks.
  static func buildBlock(_ components: Component...) -> Component { ... }

  /// If declared, provides contextual type information for statement
  /// expressions to translate them into partial results.
  static func buildExpression(_ expression: Expression) -> Component { ... }

  /// Enables support for `if` statements that do not have an `else`.
  static func buildOptional(_ component: Component?) -> Component { ... }

  /// With buildEither(second:), enables support for 'if-else' and 'switch'
  /// statements by folding conditional results into a single result.
  static func buildEither(first component: Component) -> Component { ... }

  /// With buildEither(first:), enables support for 'if-else' and 'switch'
  /// statements by folding conditional results into a single result.
  static func buildEither(second component: Component) -> Component { ... }

  /// Enables support for..in loops in a result builder by combining the
  /// results of all iterations into a single result.
  static func buildArray(_ components: [Component]) -> Component { ... }

  /// If declared, this will be called on the partial result of an 'if
  /// #available' block to allow the result builder to erase type
```

```
  /// information.
  static func buildLimitedAvailability(_ component: Component) -> Component { ... }

  /// If declared, this will be called on the partial result from the outermost
  /// block statement to produce the final returned result.
  static func buildFinalResult(_ component: Component) -> Result { ... }
}
```