Structures of Patterns

Overview of `match`

```
expr match pattern
```

```
expr match {
   pattern1 => statement1
   pattern2 => statement2
   ...
}
```

```
if (expr match pattern) {
   ...
}
```

```
auto id match pattern = init;
```

Overview of Patterns

```
expr match {
  42     => // is 42
  a      => // is a
  let x => // x is _

  int      => // is int
  concept => // is concept

  ? pattern => // * pattern
}
```

```
expr match {
  [0, a]      => // is [0, a]
  let [x, y] => // [x, y] is _

  <int> 42 => // as int is 42
  <concept> let x => // N/A?

  let x && 7 => // x is 7
  [0, let x] || [let x, 0] =>
    // [_, x] is [0, _] ||
    // [x, _] is [_, 0]
}
```

Pattern Composition

```
struct Rgb { int r, g, b; };
struct Hsv { int h, s, v; };

using Color = variant<Rgb, Hsv>;

struct Quit {};
struct Move { int x, y; };
struct Write { string s; };
struct ChangeColor { Color c; };

using Command = variant<Quit, Move, Write, ChangeColor>;

Color color = Rgb { 0, 160, 255 };
Command command = ChangeColor { c };
```

Pattern Composition

```
color match {
  <Rgb> let [r, g, b] => ...
};
```

```
inspect (color) {
  [r, g, b] as Rgb => ...
}
```

Pattern Composition

```
color match {
  <Rgb> let [r, g, b] => ...
};

command match {
  <ChangeColor> [
    <Rgb> let [r, g, b]
  ] => ...
};
```

```
inspect (color) {
  [r, g, b] as Rgb => ...
}

inspect (command) {
  [[r, g, b]] as ChangeColor
                 as [Rgb]
    => ...
}
```

Pattern Composition

```
color match {
  <Rgb> let [r, g, b] => ...
};

command match {
  <ChangeColor> [
    <Rgb> let [r, g, b]
  ] => ...
};
```

```
inspect (color) {
  [r, g, b] as Rgb => ...
}

inspect (command) {
  [[r, g, b]] as ChangeColor
                 as [Rgb]
    => ...
}
```

Pattern Composition

```
color match {
  <Rgb> let [r, g, b] => ...
};

command match {
  <ChangeColor> [
    <Rgb> let [r, g, b]
  ] => ...
};
```

```
inspect (color) {
  [r, g, b] as Rgb => ...
}

inspect (command) {
  [[r, g, b]] as ChangeColor
                 as [Rgb]
    => ...
}
```

Sprawling and Repeated Structures

```
color match {
  [a, let y] => …
}
```

```
inspect (color) {
  [_, y] is [a, _] => …
}
```

Sprawling and Repeated Structures

```
color match {
  <Rgb> let [r, g, b] => ...
};

command match {
  <ChangeColor> [
    <Rgb> let [r, g, b]
  ] => ...
};
```

```
inspect (color) {
  [r, g, b] as Rgb

    => ...
}

inspect (command) {
  [[r, g, b]] as ChangeColor
              as [Rgb]

    => ...
}
```

Sprawling and Repeated Structures

```
color match {
  <Rgb> [0, 160, let b] => ...
};

command match {
  <ChangeColor> [
    <Rgb> [0, 160, let b]
  ] => ...
};
```

```
inspect (color) {
  [_, _, b] as Rgb
            is [0, 160, _]
    => ...
}

inspect (command) {
  [[_, _, b]] as ChangeColor
              as [Rgb]
              is [[0, 160, _]]
    => ...
}
```

Some Things Do Compose

```
p match {
  ?[_, let y] => ...
};

p match {
  [?[_, let y], ?42] => ...
};

p match {
  [?[a, let y], ?<int> 42] => ...
};
```

```
inspect (p) {
  *[_, y] is _ => ...
}

inspect (p) {
  [*[_, y], _] is [_, *42] => ...
}

inspect (p) {
  [*[_, y], _] as [_, *int]
                  is [*[a, _], 42]
    => ...
}
```

Some Things Do Compose

```
p match {
  ?[_, let y] => ...
};

p match {
  [?[_, let y], ?42] => ...
};

p match {
  [?[a, let y], ?<int> 42] => ...
};
```

```
inspect (p) {
  *[_, y] is _ => ...
}

inspect (p) {
  [*[_, y], _] is [_, *42] => ...
}

inspect (p) {
  [*[_, y], _] as [_, *int]
                  is [*[a, _], 42]
    => ...
}
```

Poll

- Do we want fully compositional patterns or semi-compositional patterns chained with is/as?