Counter-examples for P2688R5

Sergey Anisimov
Tymofii Kutlov
Vlad Serebrennikov

Example #1: `if` statement 1

C++17

```
std::tuple<int, int> f();
if (auto [a, b] = f(); a == 1) {
  a, b; // well-formed
} else {
  a, b; // well-formed
}
```

P2688R5

```
std::tuple<int, int> f();
if (f() match [1 let a, let b]) {
  a, b; // well-formed
} else {
  a, b; // unexpectedly ill-formed
}
```

Example #2: `if` statement 2

C++17

```
std::tuple<int, int> f();
bool j = should_bypass_check();
if (auto [a, b] = f(); a == 1 || j) {
  a, b; // well-formed
} else {
  a, b; // well-formed
}
```

P2688R5

```
std::tuple<int, int> f();
bool j = should_bypass_check();
if (f() match [1 let a, let b] || j) {
  a, b; // unexpectedly ill-formed
} else {
  a, b; // unexpectedly ill-formed
}
```

Example #3: Matching types 1

C++17

```cpp
template <typename T>
constexpr bool is_int(T x) {
  if constexpr (std::is_same_v<T, int>)
    return true;
  else if constexpr (std::is_same_v<T, std::variant<int>>)
    return false;
}
constexpr std::variant<int> v{42};
static_assert(is_int(v)); // expectedly fails
```

P2688R5

```cpp
constexpr std::variant<int> v{42};
// passes counter-intuitively
static_assert(v match {
  int: _ => true;
  std::variant<int>: _ => false;
});
```

Example #4: Matching types 2

Without protocol

```
template <typename>
class Widget { /*...*/ };








constexpr Widget<int> w{};

// ill-formed
std::visit(/*...*/, w);
// well-formed, doesn't pass
static_assert(w match {
  int: _ => true;
  Widget<int>: _ => false;
});
```

With protocol

```
template <typename>
class Widget { /*...*/ };

template <typename T>
struct std::variant_size<Widget<T>> { /*...*/ };

template <std::size_t I, typename T>
struct std::variant_alternative<I, Widget<T>> { /*...*/ };

constexpr Widget<int> w{};

// well-formed
std::visit(/*...*/, w);
// well-formed, passes (!)
static_assert(w match {
  int: _ => true;
  Widget<int>: _ => false;
});
```

Example #5: Matching multiple types at once

Common code

```
using Variant = std::variant<
  std::array<int, 2>,
  std::array<int, 3>,
  std::array<float, 2>
>;
constexpr Variant v;
```

C++17

```
static_assert(std::visit(
  overloaded{
    [] <size_t N> (std::array<int, N>)
      { return true; },
    [] <size_t N> (std::array<float, N>)
      { return false; }
  }, v
));
```

P2688R5

```
static_assert(v match {
  std::array<int, 2>:   _ => true;
  std::array<int, 3>:   _ => true;
  std::array<float, 2>: _ => false;
});
```

P2688R5 + concepts

```
template<typename T, typename ValueT>
concept array_of = requires (T* x) {
  [] <size_t I> (std::array<ValueT, I>*){}(x);
};

static_assert(v match {
  array_of<int>:   _ => true;
  array_of<float>: _ => false;
});
```

Example #6: Attributes

Single identifier

```
42 match {
  // invented syntax, should be fine
  0 let x [[maybe_unused]] => true;
};
```

Whole match case

```
struct A {
  int a;
};
struct B {
  A b;
};

constexpr int attr = 24;

B{} match {
  // a well-formed pattern (!)
  [[attr]] let x => true;
};
```

Additional questions

1. Interaction with parameter packs seem unexplored. Aren’t we closing future design doors by ignoring them today?