Doc. No.: P3210R0
Project: Programming Language - C++ (WG21)
Author: Andrew Tomazos <andrewtomazos@gmail.com>
Date: 2024-03-29
Audience: SG21 Contracts, Evolution
Target: C++26

A Postcondition *is* a Pattern Match

Summary

We propose (a) adopting the P2737-recommended syntax, `post(result > 0)`, for referring to the function result in a postcondition into P2900 MVP Contacts, replacing the current syntax `post(r: r > 0)`; and (b) adopting a future extension to that syntax to ship in the same standard version as the P2688 Pattern Matching `match` expression ships in. The future extension has the form `post(`**`α`**`)`, where **α** has the same syntax as the **α** in `result match {` **`α`**`; }` does.

We explain how these two changes result in a simpler and more consistent C++ language syntax.

Motivation

How important is postcondition syntax that uses the return value?

At least the majority of functions are value-returning. An overwhelming majority of value-returning functions have postconditions **about the result** of the function.

Therefore, the syntax for referring to the function result in a postcondition will be prolific, and so it is of utmost importance that we get it right.

How are postcondition predicates related to pattern matching?

A P2688 Pattern Match match expression has the form:

```
    <subject> match { <pattern-output>; }
```

It takes a subject expression and produces an output value that is **about the subject**.

The semantics of the `<pattern-output>` part of this expression are the same as that of a postcondition predicate that refers to the return value. That is, **a postcondition predicate is a pattern match about the return value**.

As per the well-established language design principle that things with similar semantics should have similar syntax, the syntax of a postcondition about the result should therefore be:

```
post(<pattern-output>)
```

What of postconditions that are not about the return value?

In many cases a postcondition is not about the return value, and is just a boolean expression. Typically, this would be a test that a function's side effects have occurred.

So we also need to maintain the existing form:

```
post(<expression>)
```

Fortunately, the two forms can be easily disambiguated during parsing.

Why are you also proposing adopting the P2737 result syntax `post(result > 0)`?

In addition to the motivation already given in P2737 we add the following two points:

1) The syntax creates a simple and conceptual bridge between the expression form and pattern-match form. This can be highlighted by specifying the P2737 syntax in the different, but importantly equivalent, fashion:

A postcondition of the form:

```
post(<expression>)
```

is equivalent to a postcondition of the form:

```
post(let result => <expression>)
```

which is a specific case of the more general:

```
post(<pattern-output>)
```

2) Adopting the P2737 `post(result > 0)` syntax enables us to merge and ship P2900 contracts before P2688, while maintaining forward-compatibility with the future extension `post(<pattern-output>)` proposed here. The current P2900 syntax `post(r: r > 0)` is

inconsistent with the P2688 syntax `post(let r => r > 0)`, hence why we are motivated to replace it with `post(result > 0)`.

Examples

Generally examples of the syntax of the future extension can be generated by taking any P2688 example and replacing:

```
   α match { β; }
```

with

```
  post(β)
```

(ie **α** is the return value.)

```
// ex. 1: void-returning postconditions
int global;

void f()
  post(global == 42);  // P2900 unchanged

// ex. 2: simple postcondition of value-returning function
int f()
  post(result > 0);  // P2737

// ex. 3: complex postcondition of value-returning function
float f()
  post(let r => r*r*r + 2*r*r - 3*r + 4);

// ex. 4: postcondition on integer
int f()
  post(
    0 => default_available();
    1 => true;
    _ => false);

// ex. 5: postcondition on string
std::string f()
  post(
    "foo" => false;
    "bar" => true;
    let s => is_zipcode(s));
```

```
// ex. 6: postcondition on tuple (structured binding)
tuple<int,int> f()
  post(
    [0, 0] => true;
    [0, let y] => y < 10;
    [let x, 0] => x < 20;
    let [x, y] => x + y < 4);

// ex. 7: postcondition on variant
variant<int32_t, int64_t, float, double> f()
  post(
    int32_t: let i32 => i32 < (1 << 30);
    int64_t: let i64 => i64 < (1ll << 60);
    float: let fl => fl < 1.0e48;
    double: let d => d < 1.0e96);

// ex. 8: postcondition on concept
template<typename T>
T f()
  post(
    std::integral: let i => i < 100;
    std::floating_point: let f => f < 1.0;
    _: false);

// ex. 9: postcondition on polymorphic type
struct Shape { virtual ~Shape() = default; };
struct Circle : Shape { int radius; };
struct Rectangle : Shape { int width, height; };

Shape& f()
  post(
    Circle: let [r] => r > 0;
    Rectangle: let [w, h] => w > 0 && h > 0);

// ex. 10: postcondition on nested structure
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>;

Command f()
  post(
    Quit: _ => quit_queued();
    Move: let [x, y] => x > y;
    Write: let [text] => !text.empty()
    ChangeColor: [Rgb: let [r, g, b]] => r == g && g == b;
    ChangeColor: [Hsv: let [h, s, v]] => s == 0);
```

Proposals

Proposal 1

In P2900 we should replace the syntax `post(r : r > 0)` with the P2737 syntax `post(result > 0)`

Proposal 2

In the same standard that P2688 Pattern Matching ships in, we should add an extension to the postcondition syntax `post(`**α**`)`, where **α** has the same syntax as it does in `result match {` **α**`; }`.

FAQ

With the P2737 syntax, what happens when the name of a parameter is `result`?

For example:

```
    int f(int result)
       post(result > 0); // ill-formed: ambiguous
```

You can rename the parameter:

```
    int f(int result_in)
       post(result > 0); // OK: return value

    int f(int result_in)
       post(result_in > 0); // OK: parameter
```

Or you can wait for the future extension and write:

```
int f(int result)
   post(let return_value => return_value > 0); // OK: return value

int f(int result)
   post(let return_value => result > 0); // OK: parameter
```

This is treated further in the FAQ section of P2737

Have you considered proposing `post(let r => r > 0)` as the replacement for `post(r: r > 0)`?

Yes, we think this depends too much on the final details of the P2688 syntax to ship it in P2900 prior to P2688 shipping. There may be minor syntax changes in P2688 that impact this syntax. `post(result > 0)` has little dependance on the fine details of the P2688 syntax, as such it is low risk to ship ahead of P2688 in P2900.

In the unlikely event that the final P2688 syntax is ambiguous with an expression, we have a backup plan of shipping the P2688 syntax as `post{let r => r > 0}` with the braces serving to disambiguate the expression form `post(result > 0)` from it. (In such a case this disambiguation would also be needed regardless of the P2737 syntax anyway.)

Acknowledgements

Thank you to Ran Reglev for bringing the underlying issue to our attention.

Thank you to Li Yihe for spotting the key concept that ties postconditions and pattern matching.

References

[P2688] https://wg21.link/P2688
Pattern Matching: match Expression
Document #: D2688R1
Date: 2024-02-15
Project: Programming Language C++
Audience: Evolution
Reply-to: Michael Park
<mcypark@gmail.com>

[P2737] https://wg21.link/P2737
Proposal of Condition-centric Contracts Syntax
Doc. No.: P2737R0

Project: Programming Language - C++ (WG21)
Audience: SG21 Contracts
Author: Andrew Tomazos <andrewtomazos@gmail.com>
Date: 2022-12-04

[P2900] https://wg21.link/P2900
Contracts for C++
Document #: P2900R5
Date: 2024-02-15
Project: Programming Language C++
Audience: EWG, LEWG
Reply-to: Joshua Berne <jberne4@bloomberg.net>
Timur Doumler <papers@timur.audio>
Andrzej Krzemieński <akrzemi1@gmail.com>
— with —
Gašper Ažman <gasper.azman@gmail.com>
Tom Honermann <tom@honermann.net>
Lisa Lippincott <lisa.e.lippincott@gmail.com>
Jens Maurer <jens.maurer@gmx.net>
Jason Merrill <jason@redhat.com>
Ville Voutilainen <ville.voutilainen@gmail.com>