Doc. No.: P3210R2
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 (1) adopting the P2688 binding syntax `post(let r => r > 0)` into P2900 MVP Contacts, replacing the current P2900 binding syntax `post(r: r > 0)` (and keeping this binding syntax in P2900 future revisions in sync with P2688 future revisions); and (2) adopting a future generalization of that syntax to ship in the same standard version that P2688 ships in. The future generalization has the form `` `post(**β**)` ``, where **β** has the same grammar as the **β** in **α** `match { **β**; }` does. Notice that (1) is a special case of (2). That is, (2) subsumes (1).

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

Change Log

R1->R2

- Change Proposal 1 to be “match binding syntax of P2688” rather than “adopt keyword syntax of P2737”
- Updated rest of paper to match
- Expanded and clarified Motivation.

R0->R1:

- Editorial changes

Motivation

How important is the syntax of postconditions about the return value?

At least the majority of functions are value-returning (for example, ~70% of std::vector member functions are value-returning). An overwhelming majority of value-returning functions have postconditions about the return value: “The function returns blah blah blah” (for example, all value-returning member functions of std::vector have such a postcondition).

Therefore, the syntax for writing such postconditions will be prolific, and so it is of utmost importance that we get it right.

How are postcondition predicates related to pattern matching? What does the paper title “A Postcondition *is* a Pattern Match” mean?

A P2688 Pattern Match match expression has the form:

```
α match { β; }
```

The semantics of **β** is that it denotes a **value that is about the value of α.** (That is, **β** describes a value in the context of **α.**)

The semantics of a postcondition about the return value, are that it denotes a bool **value about the return value**.

In both cases the semantics are that **a value is denoted about a subject value**.

That’s what we mean by the title “A Postcondition is a Pattern Match”. They have the same semantics.

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

```
post(β)
```

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 non-pure function’s effects have occurred.

So we also need to maintain the existing expression syntax:

```
post(expression)
```

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

Examples

Generally examples of the syntax of the future generalization 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(let result => result > 0);

// ex. 3: complex postcondition of value-returning function
float f()
  post(let r => r*r*r + 2*r*r - 3*r + 4 > 0);

// ex. 4: decomposition pattern (like structured binding)
tuple<A,B> f()
  post(let [a,b] => is_cotangled(a,b))

// ex. 5: postcondition on integer
int f()
  post(
    0 => default_available();
    1 => true;
    _ => false);

// ex. 6: postcondition on string
std::string f()
  post(
    “foo” => false;
    “bar” => true;
    let s => is_zipcode(s));

// ex. 7: complex 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. 8: 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. 9: 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. 10: 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. 11: 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 binding syntax `post(r : r > 0)` with the current P2688 binding syntax `post(let r => r > 0)`, and we should keep it in sync with future revisions of P2688 until P2900 ships in a release vehicle.

In P2688: after P2900 ships in an ISO release vehicle, we should not change that part of the P2688 syntax.

Proposal 2

In the same standard that P2688 Pattern Matching ships in, we should add an extension to the postcondition syntax post(**β**), where **β** has the same syntax as it does in **α** match { **β**; }. (Proposal 1 becomes a special case of Proposal 2.)

Acknowledgements

Thank you to Ran Regev 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>

[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>