Is a parameter type with `T&&` in it and `T` deduced always a universal reference?

Asked 1 year, 4 months ago Modified 1 year, 4 months ago Viewed 93 times

With reference to

```
template<typename T>
void fun(ParamType param); // ParamType is some form of T, e.g. T, T&, T const, ...

fun(expr); // expr is an expression
```

I know that `&&` "somewhere" in `ParamType` is not enough for `ParamType` to be a universal reference, for instance `ParamType = std::vector<T>&&` is an rvalue reference. `&&` needs to be just beside the template parameter `T` for it (`T&&`) to *possibly* be a universal reference.

But I also know that `T&&` "somewhere" in `ParamType` is still not enough, for instance `ParamType = std::remove_reference_t<T&&>` is not a universal reference.

Therefore, I was almost asking *are there/what are the requirements on the "form" of `ParamType` for it to be a forwarding reference?*.

However, by doing a bit of research, I kind of feel I need to ask a more "binary" question:

Assuming for simplicity that `T` is the only template parameter and, consequently, that `ParamType` is only function (in the math sense) of `T`, is it correct to say that if both of these are true

- `T` appears in `ParamType` the form of `T&&`
- `fun` deduces `T` and `PramType` (i.e. the `T` doesn't need to be passed via `fun<T>`)

then `ParamType` is a universal reference?

In other words, is `ParamType` always a universal reference if it has a `T&&` in it with `T` deduced?

c++ c++11 template-meta-programming rvalue-reference forwarding-reference

Share Edit Follow Flag

asked Nov 22, 2020 at 16:34
Enlico
17.2k 5 36 74

2 Answers

Sorted by: Highest score (default)

> what are the requirements on the "form" of ParamType for it to be a forwarding reference?.

From the standard (latest draft):

> [temp.deduct.call] A forwarding reference is an rvalue reference to a cv-unqualified template parameter that does not represent a template parameter of a class template

> Is a parameter type with `T&&` in it and `T` deduced always a universal reference?

It doesn't matter what is "in" the type, but what the type **is**. If it is a cv-unqualified rvalue reference to a template parameter of a function, then it is a forwarding reference.

For example, `Foo<T&&>` has `T&&` "in it", but it is not a forwarding reference. Nor can `ParamType` be a forwarding reference since it isn't a template argument of the function.

Share Edit Follow Flag

edited Nov 22, 2020 at 17:36

answered Nov 22, 2020 at 16:38
eerorika
218k 11 172 290

Mmm, this helps a lot, but I'm starting to think that there's no way to have a universal reference if `ParamType` is not just `T&&`. Is this correct? – Enlico Nov 22, 2020 at 17:25

@Enrico That's the only way to have a forwarding reference. – eerorika Nov 22, 2020 at 17:27

That's the only way to have a forwarding reference. Does it mean *that's the only way it **can** be a forwarding reference* or *that's the only way it **can** be **and is** a forwarding reference*? Based on the excerpt you quoted from the standard, it looks like it's the latter case. – Enlico Nov 22, 2020 at 17:36

1 @Enrico It stops being a forwarding reference if `T` (the thing to the left of `&&`) is not being deduced when the call is made, e.g. because you've manually specifed the template argument. – HolyBlackCat Nov 22, 2020 at 17:40

1 @Enrico Depending on reasons why you want argument to not be a forwarding ref, solutions to the problem may be an lvalue reference overload which would be preferred over the universal reference overload, or `std::enable_if` based SFINAE that prevents the function from being valid for lvalues. – eerorika Nov 22, 2020 at 18:23

|

No, it isn't.

I did not think of the obvious case!

```
template<typename T>
void fun(const T&&) {}
```

Here `T&&` is part of `ParamType` (which is `const T&&`), `T` is deduced, but `ParamType` is an rvalue reference.

(Probably I did not think of this because I picked the habit of putting `const` after the type, as in `T const&&`.)

Share Edit Follow Flag

edited Nov 22, 2020 at 16:40

answered Nov 22, 2020 at 16:37
Enlico
17.2k 5 36 74

1 note that "universal reference" is a term coined by Scott Meyers and it was later given the official name forwarding references. Scott did a great job at realizing that they need a name at all. His article might help: isocpp.org/blog/2012/11/... – 463035818_is_not_a_number Nov 22, 2020 at 16:55

Yeah, I'm reading that book for the second or third time and writing some summaries for myself, that's why I asked this question, which I tagged as *forwarding-reference* for the reason you mention. Your comment can be certainly be useful for the reader. – Enlico Nov 22, 2020 at 17:23