`std::execution::decay_copy`

Document Number: P4293R0
Date: 2026-07-15
Reply-to: Robert Leahy <rleahy@rleahy.ca>
Audience: SG1, LEWG

Abstract

This paper proposes an asynchronous algorithm which decay-copies result datums sent via a by-reference completion signature parameter.

Background

As adopted into C++26 `std::execution` [1]:

- Allows for asynchronous operations which advertise ref-qualified completions, and
- Contains many algorithms which decay-copy result datums regardless of whether or not those result datums correspond to an advertised ref-qualified completion parameter

P4288 [2] proposes changing the second bullet above. In the course of arguing for such a change it posits:

“Decay-copying [...] can be encapsulated in a distinct algorithm which addresses that single purpose. That algorithm can then be combined with [other algorithms] when [...] required.”

This paper aims to prove the above by proposing said “distinct algorithm” which “[d]ecay-cop[ies].”

Discussion

Needfulness

If decay-copying of reference completions is not needful then P4288’s central thesis is strengthened (since it objects to the eager manner in which `std::execution` algorithms decay-copy). Since there are those who aren’t in full agreement with the central thesis of P4288 it follows that this algorithm may be needful.

Functionality

As discussed in P4288 there are three types of completion signature:

- Value
- Rvalue reference
- Lvalue reference

P4288 discusses at length the manner in which a completion signature corresponds to an actual completion which is sent, including discussing the fact that receiving an rvalue completion may indicate that the corresponding signature was by value, or by rvalue reference.

If a “result datum” (§33.3 [exec.async.ops]) is an rvalue which corresponds to a by-value parameter of a completion then decay-copying it is needless. The fact it is reified as an rvalue reference is simply an optimization (as discussed in P4288) and it will be persisted in some manner if needed downstream.

Therefore only result datums which correspond to a by-reference parameter of a completion ought to be decay-copied and then re-sent. Note that in the case of by-rvalue-reference completion signatures it is sufficient to simply change the advertised completion signature type (since both by-value and by-rvalue-reference completions are received by rvalue reference, as detailed by P4288).

Proposal

[execution.syn]

```
namespace std::execution {
  [...]
  // [exec.adapt], sender adaptors
  template<class-type D>
    struct sender_adaptor_closure { };

  struct starts_on_t { unspecified };
  struct continues_on_t { unspecified };
  struct on_t { unspecified };
  struct schedule_from_t { unspecified };
  struct then_t { unspecified };
  struct upon_error_t { unspecified };
  struct upon_stopped_t { unspecified };
  struct let_value_t { unspecified };
  struct let_error_t { unspecified };
```

```
  struct let_stopped_t { unspecified };
  struct bulk_t { unspecified };
  struct bulk_chunked_t { unspecified };
  struct bulk_unchunked_t { unspecified };
  struct when_all_t { unspecified };
  struct when_all_with_variant_t { unspecified };
  struct into_variant_t { unspecified };
  struct stopped_as_optional_t { unspecified };
  struct stopped_as_error_t { unspecified };
  struct associate_t { unspecified };
  struct spawn_future_t { unspecified };
  struct decay_copy_t { unspecified };

  inline constexpr unspecified write_env{};
  inline constexpr unspecified unstoppable{};
  inline constexpr starts_on_t starts_on{};
  inline constexpr continues_on_t continues_on{};
  inline constexpr on_t on{};
  inline constexpr schedule_from_t schedule_from{};
  inline constexpr then_t then{};
  inline constexpr upon_error_t upon_error{};
  inline constexpr upon_stopped_t upon_stopped{};
  inline constexpr let_value_t let_value{};
  inline constexpr let_error_t let_error{};
  inline constexpr let_stopped_t let_stopped{};
  inline constexpr bulk_t bulk{};
  inline constexpr bulk_chunked_t bulk_chunked{};
  inline constexpr bulk_unchunked_t bulk_unchunked{};
  inline constexpr when_all_t when_all{};
  inline constexpr when_all_with_variant_t when_all_with_variant{};
  inline constexpr into_variant_t into_variant{};
  inline constexpr stopped_as_optional_t stopped_as_optional{};
  inline constexpr stopped_as_error_t stopped_as_error{};
  inline constexpr associate_t associate{};
  inline constexpr spawn_future_t spawn_future{};
  inline constexpr decay_copy_t decay_copy{};
}
```

[exec.decay.copy]

Note: This is a new section.

Note: The below supposes the adoption of P4288 via use of `std::execution::storage_for_completion_signatures`.

`decay_copy` adapts a single input sender into a sender which decay-copies result datums generated by its child operation.

The name `decay_copy` denotes a customization point object. For subexpression `sndr`, if `decltype((sndr))` does not satisfy `sender`, `decay_copy(sndr)` is ill-formed.

Otherwise, the expression `decay_copy(sndr)` is expression-equivalent to *`make-sender`*`(decay_copy, type_identity<decltype(sndr)>{}, sndr)`.

The exposition-only class template *`impls-for`* ([exec.snd.expos]) is specialized for `decay_copy_t` as follows:

```
namespace std::execution {
  template<>
    struct impls-for<decay_copy_t> : default-impls {
      static constexpr auto complete = see below;

      template<class Sndr, class... Env>
        static consteval void check-types() {
          auto cs = get_completion_signatures<Sndr, Env...>();
          storage_for_completion_signatures<decltype(cs)>::
            get_completion_signatures();
        }
    };
}
```

The member *`impls-for`*`<decay_copy_t>::`*`complete`* is initialized with a callable object equivalent to the following lambda expression:

```
[]<class Rcvr, class Sndr, class Tag, class... Args>(
  auto, type_identity<Sndr>, Rcvr& rcvr, Tag, Args&&... args) noexcept
{
  using signature = matching_completion_signature_t<
    completion_signatures_of_t<Sndr, env_of_t<Rcvr>>,
    Tag,
    Args...>;
  COMPLETE<signature>(rcvr, Tag{}, std::forward<Args>(args)...);
}
```

For a function type `Tag(Types...)`, let *`COMPLETE`*`<Tag(Types...)>(rcvr, tag, args...)` be equivalent to:

```
TRY-EVAL(
  rcvr,
  Tag()(
    std::move(rcvr),
    DECAY-ARG<Types>(std::forward<Args>(args))...));
```

where `Args` is the pack of types `decltype(args)...`, and, for a type `T` and an expression `arg`, `DECAY-ARG<T>(arg)` is expression-equivalent to:

```
arg
```

if `T` is not a reference type, and otherwise is expression-equivalent to:

```
auto(arg)
```

References

[1] M. Dominiak et al. std::execution P2300R10
[2] R. Leahy. Stop the Decay P4288R0