function_ref in the wild

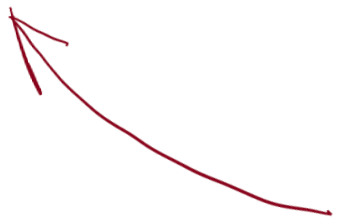
Zhihao Yuan, 2022/2/15

Recall a typical use case for function_ref

```
auto retry(function_ref<payload()> action) -> payload;
```

…

```
auto result = retry(download);
```



a function

Still a parameter

```
auto retry(retry_options opt) -> payload;
```

…

```
auto result = retry({.action = download,
                     .step_back = 1.5s});
```

More preparation

```
auto retry(retry_options opt) -> payload;
```

…

```
auto opt = default_strategy();
opt.action = download;
auto result = retry(opt);
```

Is this legal?

```
auto retry(retry_options opt) -> payload;
```

…

```
auto opt = default_strategy();
opt.action = &download;
auto result = retry(opt);
```

Creating a dangling pointer to a function ptr

```
template<class F>
function_ref(F &&f) noexcept
    : obj_(/* some cast */ addressof(f))
```

…

Back to the example

```
auto retry(retry_options opt) -> payload;
```

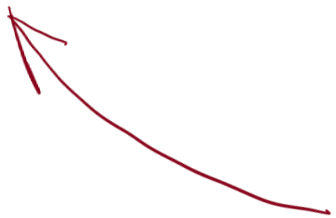
…

```
auto opt = default_strategy();
opt.action = &download;
auto result = retry(opt);
```

If that behaves undefined…

```
auto retry(retry_options opt) -> payload;
…

auto opt = default_strategy();
opt.action = ssh.get_download_callback();
auto result = retry(opt);
```



returns a function pointer

If that behaves undefined…

- Instead of writing

```
opt.action = ssh.get_download_callback();
```

- You will have to write

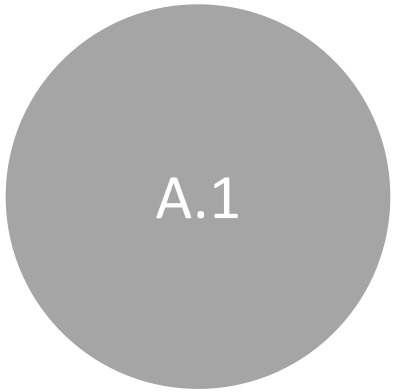
```
opt.action = *ssh.get_download_callback();
```

function_ref implementations

- `llvm::function_ref`
- `tl::function_ref`
- `folly::FunctionRef`
- `gdb::function_view`
- `type_safe::function_ref`
- `absl::function_ref`

Behavior A.1: Stores a function pointer if initialized from a function, stores a pointer to function pointer if initialized from a function pointer

```
opt.action = download; // ok
opt.action = fp;       // UB
```



- `llvm::function_ref`
- `tl::function_ref`

Behavior A.2: Stores a function pointer if initialized from a function or a function pointer

```
opt.action = download; // ok
opt.action = fp;       // ok
```

- `folly::FunctionRef`
- `gdb::function_view`
- `type_safe::function_ref`
- `absl::function_ref`

“ reference_wrapper takes addressof its argument, function_ref should also take addressof its argument

reference_wrapper – is not “one” type

```
int f();
reference_wrapper<int()> r = f;

auto fp = f;
reference_wrapper<int (*)()> r = fp;

reference_wrapper<int()> r = fp;     // nope
reference_wrapper<int (*)()> r = f; // nope
```

function_ref in question is “one” type

```
int f();
function_ref<int()> fr = f;

auto fp = f;
function_ref<int()> fr = fp;
```

Proposal

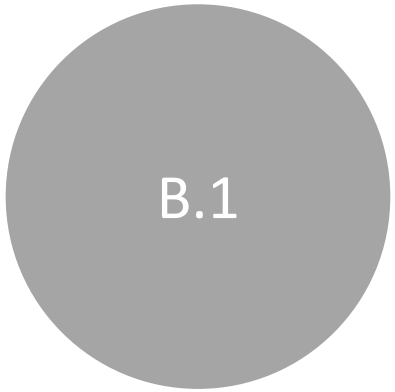
Eliminate the difference between initializing function_ref from a function and initializing function_ref from a function pointer.

A related question

- What happens when initialized from pointer-to-members?

```
function_ref<void(Ssh&)> cmd = &Ssh::connect;
```

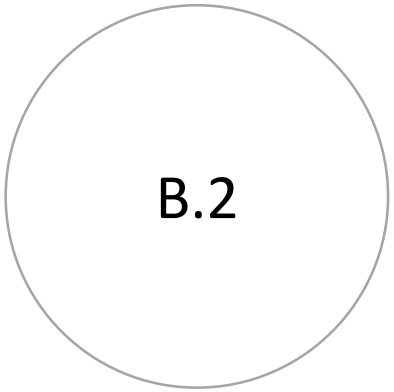
Behavior B.1: Stores a pointer to pointer-to-member if initialized from a pointer-to-member



- `tl::function_ref`
- `folly::FunctionRef`
- `absl::function_ref`

```
lib.send_cmd(&Ssh::connect); // ok
function_ref<void(Ssh&)> cmd = &Ssh::connect; // UB
```

Behavior B.2: Only supports callable entities with function call expression



- `gdb::function_view`
- `type_safe::function_ref`
- `llvm::function_ref`

```
lib.send_cmd(&Ssh::connect); // ill-formed
function_ref<void(Ssh&)> cmd = &Ssh::connect; // ill-formed
```

Arguments for ill-formed

- `std::mem_fn` is there
- Does not have to use `invoke_r`, better debug codegen
- `P2472R1`, `P2511`:

```
function_ref<void(Ssh&)> cmd = nontype<&Ssh::connect>;
```

  - Will not dangle

The cost of making function_ref bigger

```
call void @_Z3foo10TwoPointer(void ()* nonnull @_Z1fv, i8*
nonnull %3)

call void @_Z3bar12ThreePointer(%struct.ThreePointer* nonnull
byval(%struct.ThreePointer) align 8 %2)
```