Планирование задач с монады продолжения

C++ Россия, Москва 2015

Иван Чукић

ivan.cukic@kde.org
ivan.fomentgroup.org/blog

Task scheduling with the continuation monad

C++ Russia, Moscow 2015

Ivan Čukić

ivan.cukic@kde.org
ivan.fomentgroup.org/blog

About me

- KDE development
- Talks and teaching
- Functional programming enthusiast, but not a purist

Disclaimer

> Make your code readable. Pretend the next person who looks at your code is a psychopath and they know where you live.
>
> Philip Wadler

Disclaimer

The code snippets are optimized for presentation, it is not production-ready code.

`std` namespace is omitted, value arguments used instead of const-refs or forwarding refs, etc.

Moscow

MONADS

As containers

List

Maybe/Optional

Monads as containers

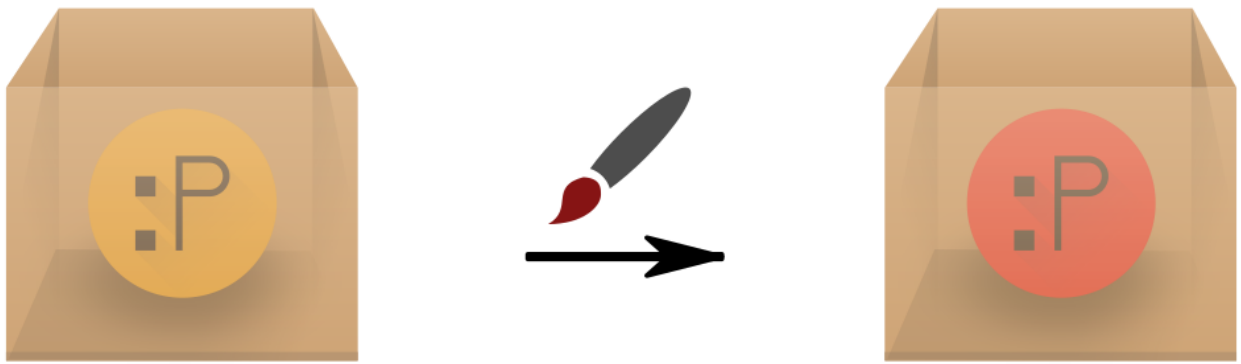
Constructor method that returns a container containing that element

```
(T) -> C<T>
```

Monads as containers

Transform (map) method

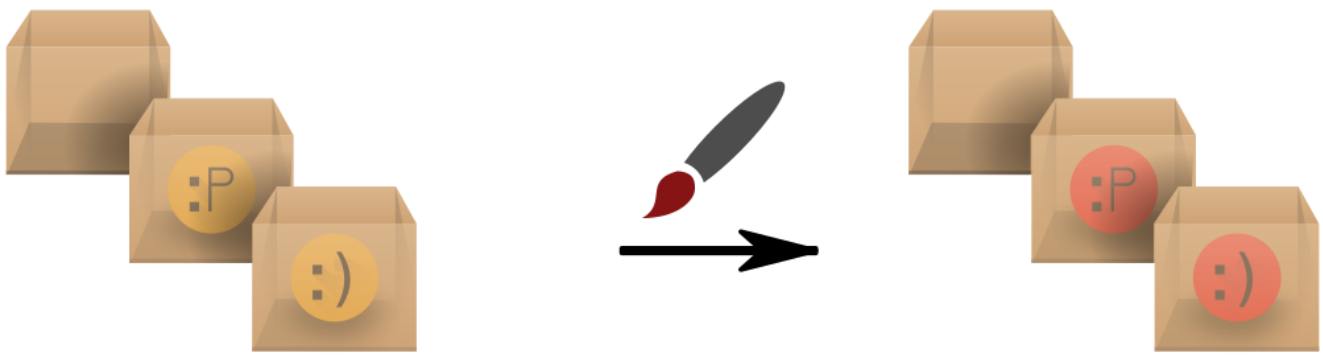
```
(C<From>, function<To(From)>) -> C<To>
```



Monads as containers

Transform (map) method

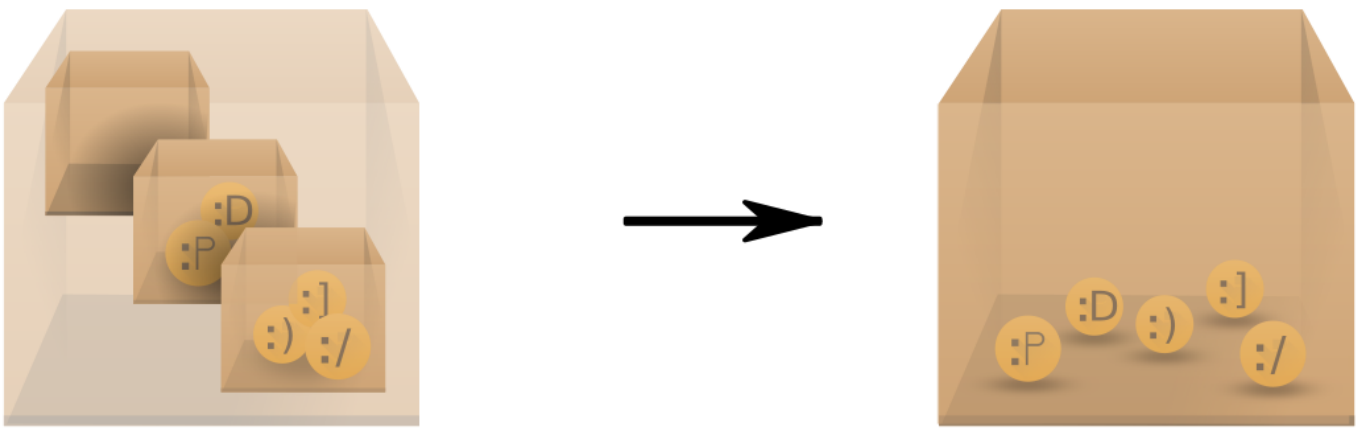
```
(C<From>, function<To(From)>) -> C<To>
```



Monads as containers

Flatten method

```
(C<C<T>>) -> C<T>
```



Monads as containers

A container-like structure with a few methods defined on it.

- Constructor method that returns a container containing that element

  ```
  (T) -> C<T>
  ```

- Transform (map) method

  ```
  (C<From>, function<To(From)>) -> C<To>
  ```

- Flatten method

  ```
  (C<C<T>>) -> C<T>
  ```

or

- Bind method

  ```
  (C<From>, function<C<To>(From)>) -> C<To>
  ```

List

```
for (item: items) {
    // do something
}

for_each(items, [] (item i) {
    // do something
});
```

List

```
auto get_young_children(list<person> people, int offset, int count)
{
    list<person> result;
    int skipped = 0, took = 0;

    for (person: people) {
        if (is_female(person))
            continue;

        list<person> children;

        for (child: get_children(person)) {
            if (child.age < 20) {
                if (skipped < offset) {
                    skipped++;
                } else {
                    :::
                }
            }
        }

        copy(children.cbegin(), children.cend(), back_inserter(result));
    }

    return result;
}
```

List, boost.range, N4128

```
auto get_young_children(list<person> people,
                        int offset, int count)
{
    return people | filtered(is_female)
                  | transformed(get_children)
                  | accumulated()
                  | filtered(is_younger_than(20))
                  | drop(offset)
                  | take(count);
}
```

Maybe, boost.optional, N3690

```
string get_query_limit() {
    auto config_limit =
        config_value("query_limit");

    if (!config_limit) return string();

    auto limit_option =
        parse_int(config_limit);

    if (!limit_option) return string();

    int limit = 1.5 * limit_option.get();

    return " LIMIT " + to_string(limit);
}
```

Maybe, boost.optional, N3690

```
string get_query_limit() {
    return (config_value("query_limit")
        | bind(parse_int)
        | transformed(1.5 * _)
        | transformed(to_string)
        | transformed(" LIMIT " + _)
    ).get_value_or("");
}
```

FUTURES

Future

Reactive streams

std::future<T>, boost.future<T>, QFuture<T>

Container for a result of an asynchonous operation.

```
auto futureResult = async(:::);

// Getting the result synchronously
futureResult.get();
```

Effects: wait()s until the shared state is ready, then retrieves the value stored in the shared state.

---

§30.6.6.15 [futures.unique_future]
C++14 Final Draft, N3936

std::future<T>, boost.future<T>, QFuture<T>

N3558, Boost.Thread $\geq$ 1.55

```
auto futureResult = async(:::);

// Getting the result asynchronously
futureResult.then([] (auto f) {
        // Safe and sound call to .get()
        f.get();
    });
```

Reactive streams

- `value -> future`
- `list/vector -> ???`

Reactive streams

Container for results of a series of asynchonous operations.

- Mouse coordinates
- Client requests in a web server
- Server response chunks

Reactive streams

```
for (item: items) {
    // do something
}

for_each(items, [] (item i) {
    // do something
});
```

Reactive streams

```
// The usual call-callback approach
void on_mouse_move(point mouse) {
     :::
}

gui_mouse_move_func(on_mouse_move);

// Reactive streams
for_each(mouse_movements, on_mouse_move);
```

Reactive streams

```
for_each(
    mouse_movements
        | filtered(scene_contains)
        | transformed(invert_y_coordinate)
    , on_mouse_move);
```

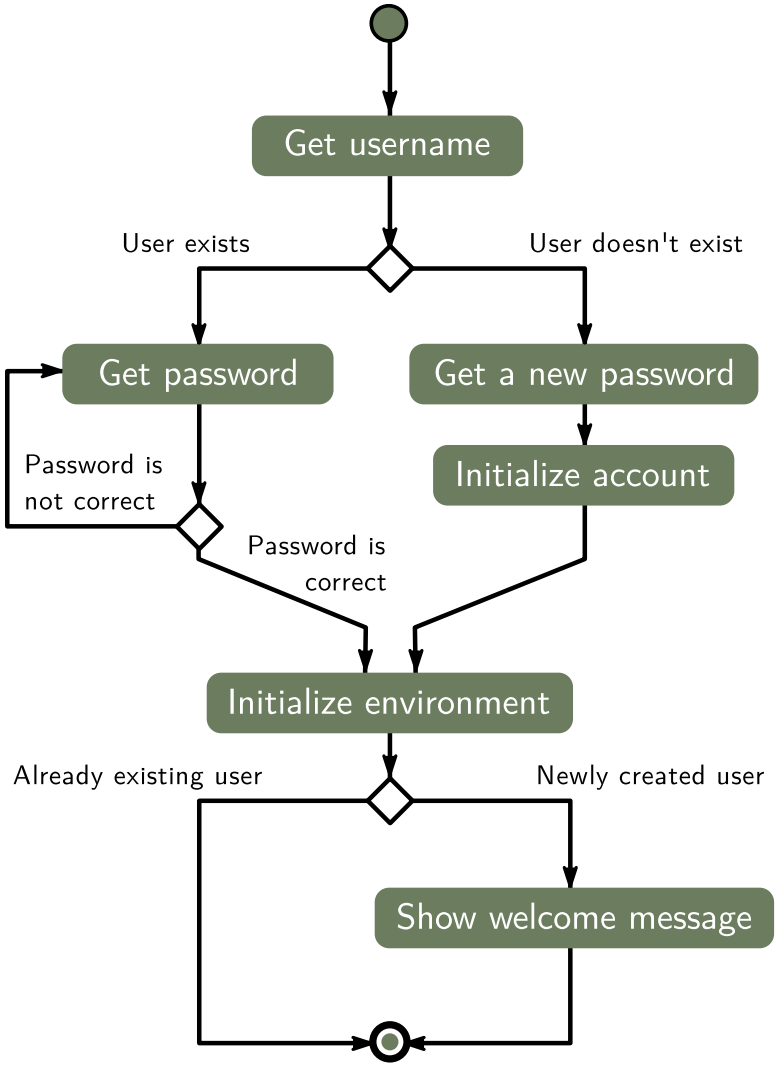
CONTINUATIONS

The problem

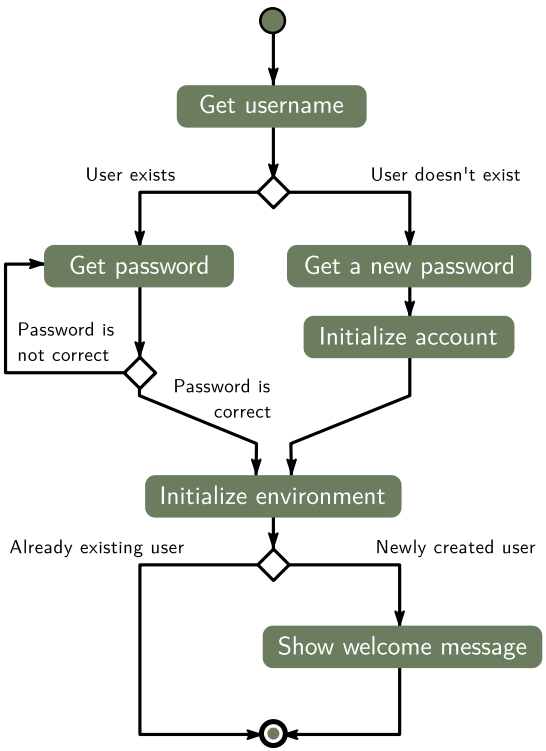
Schedulers

Set Your Controls for the Heart of the Sun

The problem


Get username
User exists
User doesn't exist
Get password
Get a new password
Password is not correct
Initialize account
Password is correct
Initialize environment
Already existing user
Newly created user
Show welcome message

The problem

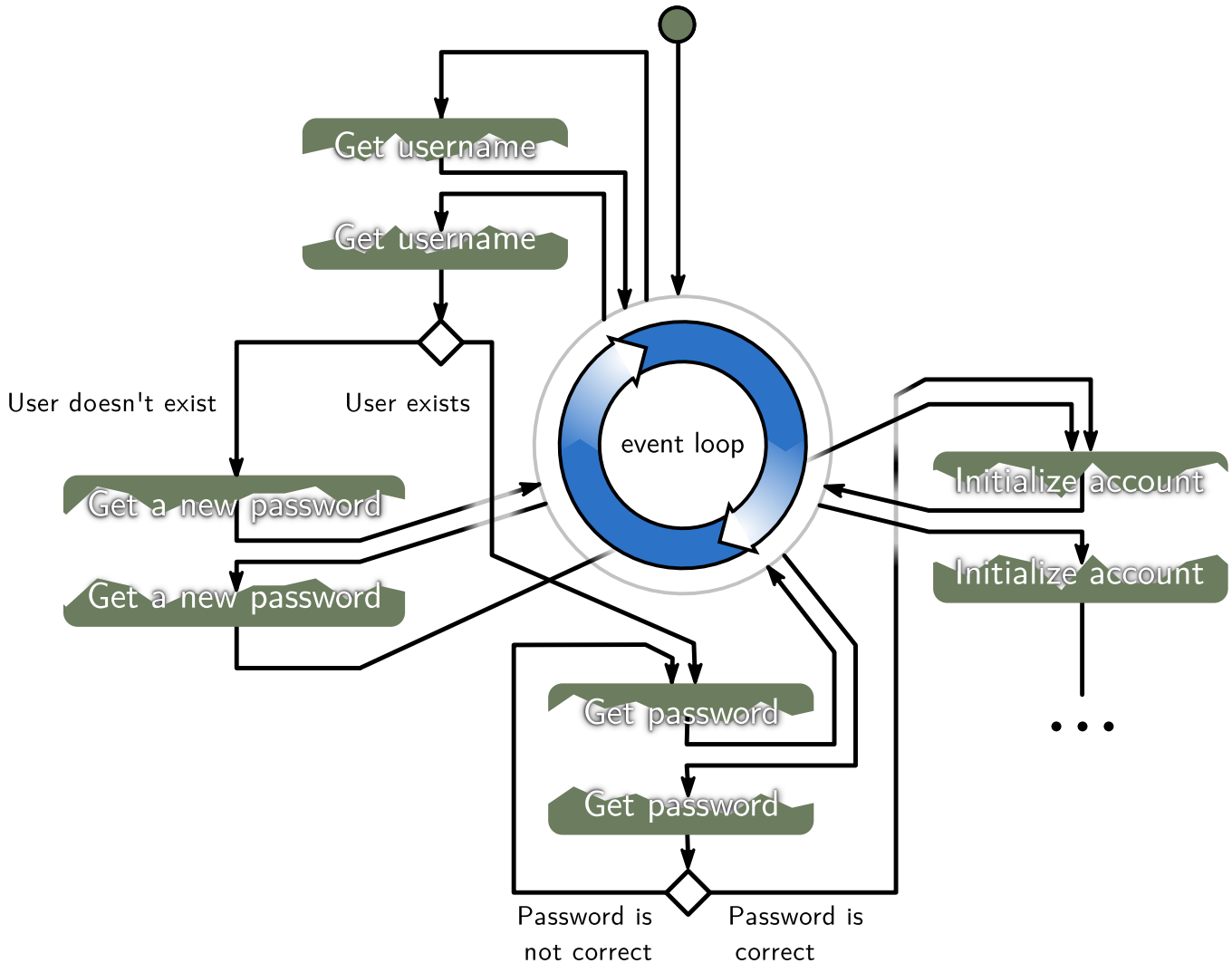



```
void login()
{
    user = get_username();

    new_user = !check_if_user_exists(user);

    if (new_user) {
        pass = get_password();
        initialize_account(uame, pass);

    } else do {
        pass = get_password();

    } while (!check_user(user, pass));

    initialize_environment();

    if (new_user) show_welcome_message();
}
```

The problem

```
void login() { get_username(on_got_username); }

void on_got_username( ::: ) {
    new_user = !check_if_user_exists(user);
    if (new_user) {
        get_password(on_got_password);
    } else { ::: }
}

void on_got_password( ::: ) {
    check_user(user, password, on_user_checked);
}

void on_user_checked( ::: ) {
    if (!user_valid) {
        on_got_username(user);
    } else {
        initialize_environment(on_environment_initialized);
    }
}

:::
```

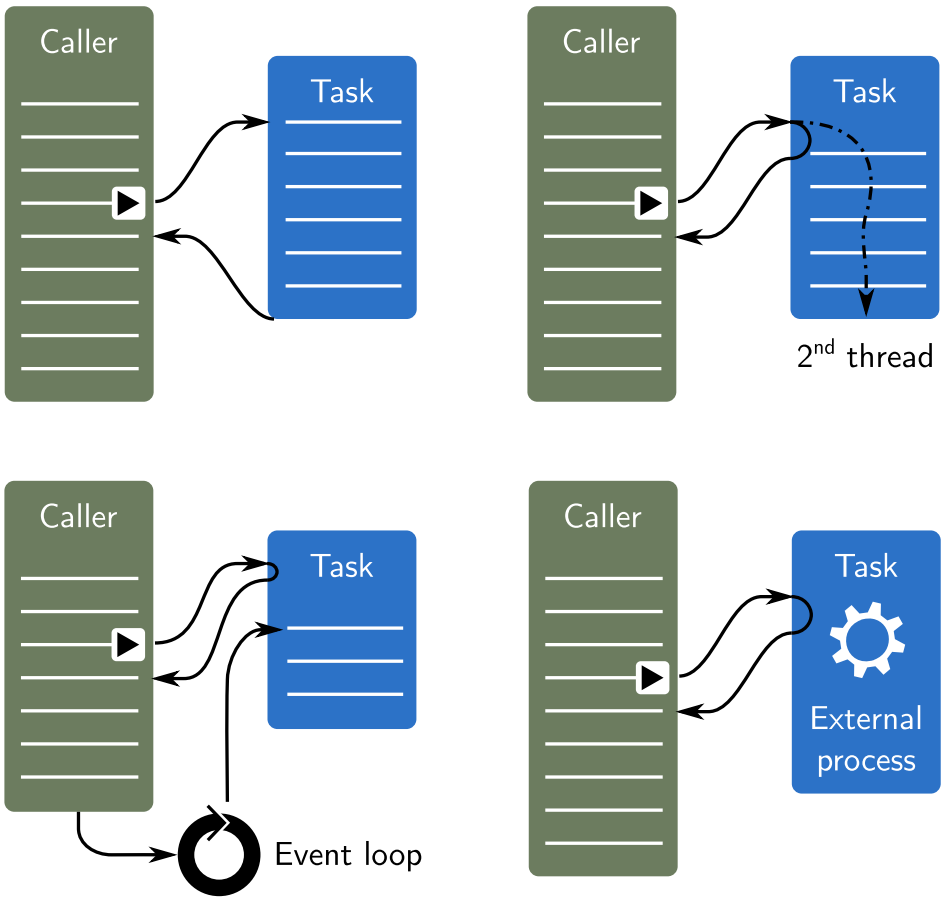
Inversion of Control



Inversion of Control

"Spaghetti code" by George W. Hart

Reasons for Waiting



- User input
- Network actions
- Inter-process communication
- External process execution
- Communication with a slow database
- CPU-intensive work
- Heterogeneous computing

...

Hiding it all, take 1

- Wrapping it in task objects (`QThread`, `KJob`, …)
- Methods with time-outs (`select`, …)
- … or with validity checks (`QProcess::state`, …)
- Future values (`future<T>`, `QFuture<T>`, `QDBusPendingReply<T>`, …)

Hiding it all, take 2

N3558, Boost.Thread $\geq$ 1.55

```
future<int> result = async(:::);

result.then([] (future<int> result) {
    // called when the result is available
    // call to .get() does not block here
    cout << result.get();
});
```

Lost in the Future

```
int i;                          |  i.then(c);          // ERROR!
                                |
future<int> future;             |  future.then(c);     // ok
                                |
QFuture<int> qfuture;           |  qfuture.then(c);    // ERROR!
```

Lost in the Future

```
int i;

future<int> future;

QFuture<int> qfuture;
```

```
c(i);

future.then(c);

auto watcher =
     new QFutureWatcher<int>();
QObject::connect(watcher,
    &QFutureWatcherBase::finished,
    [=] {
        c(watcher->result());
        watcher->deleteLater();
    });
watcher->setFuture(qfuture);
```

The Chains are On

```
getUsername().then(
    [] (future<string> username) {
        getPassword().then(
            [=] (future<string> password) {
                createAccount(username, password).then(
                    ...
                );
            }
        );
    }
);
```

Localized, but still not readable. Can it be made nicer?

The Chains are On

Can it be made to look like this?

```
void login()
{
    ...
    username = getUsername();
    password = getPassword();
    createAccount(username, password);
}
```

No, but ...

The Chains are On

... it could look like this:

```
auto login = serial_
(
    ...
    username = getUsername(),
    password = getPassword(),
    createAccount(username, password)
);
```

Peculiar syntax, but much more readable.

Let There be More Light

- while loop:

```
while_(condition) (
    body
)
```

- branching:

```
if_(condition) (
    then_branch
).else_(
    else_branch
)
```

Let There be More Light

- asynchronous operators

```
var<int> value;

value = 5;            // immediate assignment
value = someFuture(); // asynchronous assignment
```

- parallel execution

```
parallel_(
    task1,
    task2,
    ...
)
```

- parallel without waiting

```
detach_(task)
```

- producer-consumer

```
for_each(clients, process_client);
```

- transactions

etc.

Let There be More Light

```
operator(bool) // or start and undo

transaction_(
    task1,
    task2,
    ...
    taskn
    );
```

Set Your Controls...

```
var<int> wait;

serial_(
    test::writeMessage(0, "Starting the program"),

    wait = test::howMuchShouldIWait(7),
    test::writeMessageAsync(wait,
            "What is the answer to the Ultimate Question of Life, "
            "the Universe, and Everything?"
        ),

    while_(test::howMuchShouldIWait(0),
        test::writeMessageAsync(1, "42")
    ),

    serial_(
        test::writeMessageAsync(1, "We are going away..."),
        test::writeMessageAsync(1, "... sorry, but we have to.")
    ),

    test::writeMessage(0, "There, you have it!")
)();
```

… for the Heart of the Sun

```
while_(
    // Wait until we get a connection.
    client = ws::server::accept(server),

    // Start a detached execution path to process the client.
    detach_([] {
        var<ws::client_header> header;
        var<ws::message> message;
        var<string> server_key;

        serial_(
            // WebSocket handshake
            header = ws::client::get_header(),
            server_key = ws::server::create_key(header),
            ws::client::send_header(client, server_key),

            // Sending the initial greeting message
            ws::client::message_write(client, "Hello, I'm Echo"),

            // Connection established
            while_(
                // getting and echoing the message
                message = ws::client::message_read(client),
                ws::client::message_write(client, message)
            )
        )
    })
)
```

VERIFICATION AND TESTING

Introduction

Reactive streams

```
for_each(
    mouse_movements
        | filtered(scene_contains)
        | transformed(invert_y_coordinate)
    , on_mouse_move);
```

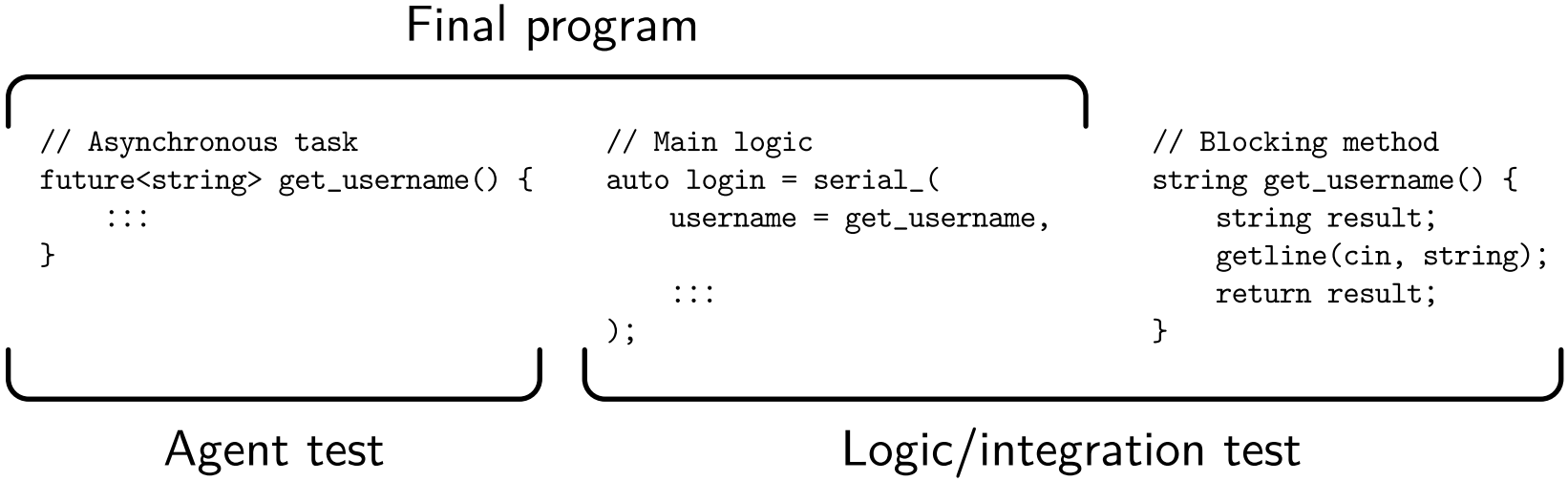
The ol' switcheroo

```
serial_(
    test::writeMessage(0, "Starting the program"),

    wait = test::howMuchShouldIWait(7),
    test::writeMessageAsync(wait,
            "What is the answer to the Ultimate Question of Life, "
            "the Universe, and Everything?"
        ),

    while_(test::howMuchShouldIWait(0),
        test::writeMessageAsync(1, "42")
    ),

    serial_(
        test::writeMessageAsync(1, "We are going away..."),
        test::writeMessageAsync(1, "... sorry, but we have to.")
    ),

    test::writeMessage(0, "There, you have it!")
)();
```

The ol' switcheroo

```
while_(
    // Wait until we get a connection.
    client = ws::server::accept(server),

    // Start a detached execution path to process the client.
    detach_([] {
        var<ws::client_header> header;
        var<ws::message> message;
        var<string> server_key;

        serial_(
            // WebSocket handshake
            header = ws::client::get_header(),
            server_key = ws::server::create_key(header),
            ws::client::send_header(client, server_key),

            // Sending the initial greeting message
            ws::client::message_write(client, "Hello, I'm Echo"),

            // Connection established
            while_(
                // getting and echoing the message
                message = ws::client::message_read(client),
                ws::client::message_write(client, message)
            )
        )
    })
)
```

Tests


Final program
// Asynchronous task
future<string> get_username() {
:::
}
// Main logic
auto login = serial_(
username = get_username,
:::
);
// Blocking method
string get_username() {
string result;
getline(cin, string);
return result;
}
Agent test
Logic/integration test

Formal code proving

- TLA+
- CAL
- Counting interface automata

Static analysis

```
Value * value = nullptr;
int i = index();

if (i >= 0) {
    assignValue(value, i)();
}

value->write();
```

Static analysis

```
int main(int argc, char *argv[])
{
    Value * value = nullptr;

    1 'value' initialized to a null pointer value →

    int i = index();

    if (i >= 0) {

        2 ← Assuming 'i' is < 0 →

    3 ← Taking false branch →

        assignValue(value, i)();
    }

    value->write();

    4 ← Called C++ object pointer is null

}
```

Static analysis

```
int main(int argc, char *argv[])
{
    Value * value = nullptr;

    int i = index();

    if (i >= 0) {

        assignValue(value, i)();
    }

    value->write();

}
```

1 'value' initialized to a null pointer value →

2 ← Assuming 'i' is < 0 →

3 ← Taking false branch →

4 ← Called C++ object pointer is null

```
auto assignValue(Value * & vp, int i,
                 void (*callback)(void))
{
    vp = getValue(i);
    callback();
}

void onAssigned() {
    value->write();
}

Value * value = nullptr;

int main(int argc, char *argv[])
{
    int i = index();

    if (i >= 0) {
        assignValue(value, i, onAssigned);
    };

}
```

Static analysis

```
int main(int argc, char *argv[])
{
    Value * value = nullptr;

    1 'value' initialized to a null pointer value →

    int i = index();

    if (i >= 0) {

        2 ← Assuming 'i' is < 0 →

    3 ← Taking false branch →

        assignValue(value, i)();
    }

    value->write();

    4 ← Called C++ object pointer is null

}
```

```
auto assignValue(Value * & vp, int i,
                 void (*callback)(void))
{
    vp = getValue(i);
    callback();
}

void onAssigned() {
    value->write();
}

Value * value = nullptr;

int main(int argc, char *argv[])
{
    int i = index();

    if (i >= 0) {
        assignValue(value, i, onAssigned);
    };

}
```

clang-analyze: **No bugs found.**

Static analysis

```
int main(int argc, char *argv[])
{
    Value * value = nullptr;

    int i = index();

    if (i >= 0) {

        assignValue(value, i)();
    }

    value->write();

}
```

1 'value' initialized to a null pointer value →

2 ← Assuming 'i' is < 0 →

3 ← Taking false branch →

4 ← Called C++ object pointer is null

```
int main(int argc, char *argv[])
{
    Value * value = nullptr;

    int i = index();

    if_ (i >= 0) (
        assignValue(value, i)
    )();

    value->write();

}
```

1 'value' initialized to a null pointer value →

2 ← Called C++ object pointer is null

AWAIT 2.0

Example

Await 2.0

- Not your father's Await
- Planned for C++17

Await 2.0

```
future<string> download(string url);

future<int> download_images(string url) {
    string html = await download(url);

    int downloaded_images_count = 0;

    for (image: get_images(html)) {
        string image_data = await download(image);
        await save_image(image, image_data);
        downloaded_images_count++;
    }

    return downloaded_images_count;
}
```

Await 2.0

> We expose tools, and library enthusiasts can go and play with them.
>
> Gor Nishanov

Await 2.0

`await expression` expands to:

```
auto && temp = expression;
if (!temp.await_ready()) {
    temp.await_suspend(...continuation...);
}

return temp.await_resume();
```

Await 2.0

```
bool await_ready(future<T> &f) {
    return f.is_ready();
}

void await_suspend(future<T> &f, C cont) {
    f.then([=](auto&){resume_callback();});
}

auto await_resume(future<T> &f) {
    return f.get();
}
```

Maybe, boost.optional, N3690

```
option<string> get_query_limit() {
    auto config_limit =
        await config_value("query_limit");

    auto limit_option =
        await parse_int(config_limit);

    int limit = 1.5 * limit_option;

    return " LIMIT " + to_string(limit);
}
```

UNDER WRAPS

Continuations

Schedulers

Under wraps

```
template <typename _Future, typename _Continuation>
void continue_with(_Future &&future,
                   _Continuation &&continuation)
{
    using is_nullary =
        typename std::is_constructible<
            std::function<void()>,
            _Continuation
        >::type;

    _continue_with_helper(
        future,
        std::forward<_Continuation>(continuation),
        is_nullary()
    );
}
```

Under wraps

```
template <typename _ReturnType, typename _Continuation>
void _continue_with_helper(const _ReturnType &value,
                           _Continuation &&continuation,
                           std::true_type)
{
    continuation();
}

template <typename _ReturnType, typename _Continuation>
void _continue_with_helper(const _ReturnType &value,
                           _Continuation &&continuation,
                           std::false_type)
{
    using is_callable = ...;
    static_assert(is_callable::value,
        "The continuation needs to at most one argument");

    continuation(value);
}
```

Under wraps

```
template <typename _ReturnType, typename _Continuation>
void _continue_with_helper(const QFuture<_ReturnType> &future,
                           _Continuation &&continuation,
                           std::false_type)
{
    if (!future.isFinished()) {
        auto watcher =
            new QFutureWatcher<_ReturnType>();

        QObject::connect(watcher, &QFutureWatcherBase::finishe
            [=] {
                continuation(watcher->result());
                watcher->deleteLater();
            });

        watcher->setFuture(future);

    } else continuation(future.result());
}
```

Matchbox

```
template<typename _TestType, typename _ArgType>
class has_then_method {
private:
    template<typename U, void (U::*)(_ArgType)>
    struct test_struct {};

    template<typename U>
    static std::true_type test(test_struct <U, &U::then> *);

    template<typename U>
    static std::false_type test(...);

public:
    using type = decltype(test<_TestType>(nullptr));
    static const bool value =
        std::is_same<type, std::true_type>::value;
}
```

The Chains are On

```
template <typename... _Jobs>
class serial_scheduler;

template <>
class serial_scheduler<> {
public:
    void operator() ()
    {
        on_end_handler(this, EXIT_SUCCESS);
    }

    void on_end(causeway::signal::handler handler) {}
};
```

The Chains are On

```
template <typename _Job, typename... _Jobs>
class serial_scheduler<_Job, _Jobs...> :
    public serial_scheduler<_Jobs...> {
private:
    using tail_t = serial_scheduler<_Jobs...>;
public:
    serial_scheduler(_Job &&job, _Jobs &&... jobs)
        : tail_t(std::forward<_Jobs>(jobs)...)
        , m_job(std::forward<_Job>(job)) {}

    void operator()() {
        auto future = this->future();

        continue_with(std::ref(m_job), [&] {
            tail_t::operator()();
        });

        return future;
    }

private:
    _Job m_job;
};
```

Answers? Questions! Questions? Answers!

Kudos:

Friends at KDE, Dr Saša Malkov, basysKom

Further reading and watching:

- Iterators Must Go, Andrei Alexandrescu
- Value Semantics and Range Algorithms, Chandler Carruth
- Systematic Error Handling in C++, Andrei Alexandrescu
- Category Theory for Programmers, Bartosz Milewski
  (expected to be awesome when released)
- Learn You a Haskell for Great Good!, Miran Lipovača
  (learning Haskell is fun and can be useful)