@odinthenerd

– not the god

This film has been modified from its original version. It has been formatted to fit your TV.

@odinthenerd

Tacit DSL all the things

狗 貓 all the things

DSL

Domain

Specific

Language

-
-
-
-

DSL

Domain

Specific

Language

- printf
-
-
-

DSL

Domain
Specific
Language

- printf
- iostreams
-
-

DSL

Domain
Specific
Language

- printf
- iostreams
- regex
-

DSL

Domain
Specific
Language

- printf
- iostreams
- regex
- ranges TS

Ranges

```
std::vector<thing> v;
for (const auto& e : v
                   | rv::reverse
                   | rv::transform(awesomeify)
                   | rv::filter(is_awesome))
{
    //do something with e
};
```

Is this good?

„Conventional programming languages are growing ever more enormous, but not stronger.

Is this good?

„Conventional programming languages are growing ever more enormous, but not stronger.

Inherent defects at the most basic level cause them to be both fat and weak:

Is this good?

„Conventional programming languages are growing ever more enormous, but not stronger.

Inherent defects at the most basic level cause them to be both fat and weak:

their primitive word-at-a-time style of programming inherited from their common ancestor --the von Neumann computer, […] and their lack of useful mathematical properties for reasoning about programs.“

Is this good?

„Conventional programming languages are growing ever more enormous, but not stronger.

Inherent defects at the most basic level cause them to be both fat and weak:

their primitive word-at-a-time style of programming inherited from their common ancestor --the von Neumann computer, [...] and their lack of useful mathematical properties for reasoning about programs.“

Is this good?

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

-- John Backus 1978

@odinthenerd

BASH

```
ls | head -3
```

BASH

```
ls | head -3
```

```
ls | head -3 | tail -1
```

Billy O'Neal
@MalwareMinigun

Antwort an @MalwareMinigun @odinthenerd und @lefticus

The goal of making easy to understand, well performing, and correct system behavior is often about minimizing the amount of mutable state that can affect the program's control flow, since mutable state that affects control flow has non-local combinatoric effects.

“what to do” parameters

```
void foo(bar& a, baz& b, bool c){
    //calculate the world
    if(c){
        //do things
    }
    else{
        //do other things
    }
}
```

```
void foo(bar& a, baz* b){
    //calculate the world
    if(b){
        //do things
    }
    else{
        //do other things
    }
}
```

```
void foo1(bar& a, baz& b){
    //calculate the world
    //do things
}
void foo2(bar& a){
    //calculate the world
    //do other things
}
```

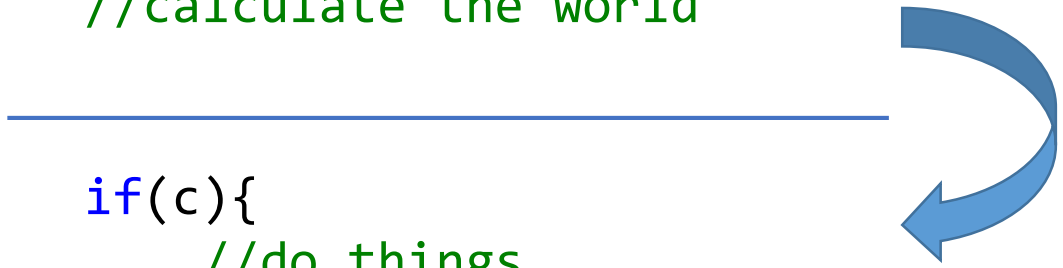
```
?? calculate_the_world(bar& a){
    //calculate the world
}
void foo1(bar& a, baz& b){
    calculate_the_world(a);
    //do things
}
void foo2(bar& a){
    calculate_the_world(a);
    //do other things
}
```

```
?? calculate_the_world(bar& a){
    //calculate the world
}
void foo1(bar& a, baz& b){
    calculate_the_world(a);
    //do things
}
void foo2(bar& a){
    calculate_the_world(a);
    //do other things
}
```

```
?? calculate_the_world(bar& a){
    //calculate the world
}
void foo1(bar& a, baz& b){
    auto [??] = calculate_the_world(a);
    //do things
}
void foo2(bar& a){
    auto [??] = calculate_the_world(a);
    //do other things
}
```

Problem of local variables

```
void foo(bar& a, baz& b, bool c){
    //calculate the world
```



```
    if(c){
        //do things
    }
    else{
        //do other things
    }
}
```

Functional solution

```
void calculate_the_world(std::function<??> f, bar& a){
    //calculate the world
    f(??);
}
```

```
void calculate_the_world(std::function<??> f, bar& a){
    //calculate the world
    f(??);
}

struct do_things{
    baz& b;
    void operator(??, bar& a){
        //do things
    }
}

void do_other_things(??, bar& a){
    //do other things
}
```

Single parameter solution

```
void do_things(int i, baz& b){
    //do things
}

void do_other_things(int i){
    //do other things
}

do_things(calculate_the_world(a),b);

do_other_things(calculate_the_world(a));
```

Can ranges help?

Can ranges help?

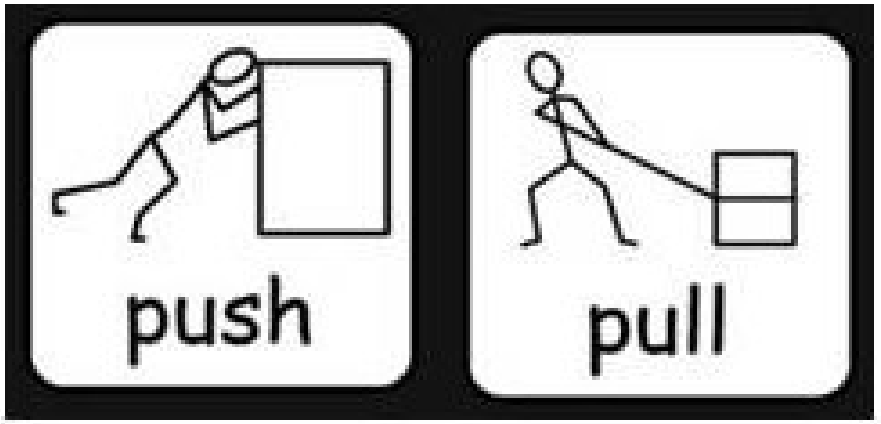
No

Can ranges help?

No

But the DSL can

Pull vs. push model



Incidental Push Model

```
void a(thing b){
    //logic b -> c,d
    f(c,d);
}
```

Incedental Push Model

```
void a(thing b){
    //logic b -> c,d
    if(c){
        f(d);
    }
    else {
        g(d);
    }
}
```

Incedental Push Model

```
void a(thing b){
     //logic b -> c,d
     if(c){
          f(d);
     }
     else {
          g(d);
     }
}
```

State Machine Here

Composing functions

```
input | fun | gun | hun

{1, 2} -> fun
```

Composing functions

```
input | fun | gun | hun
```

```
{1, 2} -> fun -> 3 -> gun
```

Composing functions

```
input | fun | gun | hun
```

```
{1, 2} -> fun -> 3 -> gun -> 6 -> hun -> void
```

Composing functions

```
input | fun | gun | hun

{1, 2} -> fun -> 3 -> gun -> 6 -> hun -> void

1 -> fun -> {1, 2, 5}
```

Composing functions

```
input | fun | gun | hun
```

```
{1, 2} -> fun -> 3 -> gun -> 6 -> hun -> void
```

```
1 -> fun -> {1, 2, 5} -> gun -> 9 -> hun -> void
```

Heterogeneous algorithms

```
auto animals = hana::make_tuple(
    Fish{"Nemo"},
    Cat{"Garfield"},
    Dog{"Snoopy"});
```

Heterogeneous algorithms

```
auto animals = hana::make_tuple(
    Fish{"Nemo"},
    Cat{"Garfield"},
    Dog{"Snoopy"});
auto get_name = [](auto& a) {
    return a.name;
};
```

Heterogeneous algorithms

```
auto animals = hana::make_tuple(
    Fish{"Nemo"},
    Cat{"Garfield"},
    Dog{"Snoopy"});
auto get_name = [](auto& a) {
    return a.name;
};
auto  names =
        hana::transform(animals, get_name) ;
```

Heterogeneous algorithms

```
auto animals = hana::make_tuple(
    Fish{"Nemo"},
    Cat{"Garfield"},
    Dog{"Snoopy"});
auto get_name = [](auto& a) {
    return a.name;
};
auto rnames = hana::reverse(
        hana::transform(animals, get_name));
```

Heterogeneous algorithms

```
auto rnames = animals |
        tmp::transform(get_name) |
        tmp::reverse;
```

```
auto rnames = hana::reverse(
        hana::transform(animals, get_name));
```

pack

- product type
- “language level tuple”-ish
- means of passing heterogeneous containers

Tacit solution

```
calculate_the_world(a) | do_other_things;
```

Tacit solution

```
calculate_the_world(a) | push_front(b) | do_things;

calculate_the_world(a) | do_other_things;
```

unwrapping

```
std::tuple<int,bool> t;
```

unwrapping

```
std::tuple<int,bool> t;

t | foo;
```

unwrapping

```
std::tuple<int,bool> t;

t >>= foo;
```

unwrapping

```
std::tuple<int,bool> t;

t >>= foo;

std::apply(foo,t);
```

unwrapping

```
std::tuple<int,bool> t;

t foo(A,B,C);

pack(a...) >>= foo | bar;
```

unwrapping

```
std::tuple<int,bool> t;

t foo(A,B,C);

pack(a...) >>= foo >>= bar;
```

Heterogeneous algorithms

```
auto rnames = animals >>=
        tmp::transform(get_name) |
        tmp::reverse;

//dsl reduces compile time


auto rnames = hana::reverse(
        hana::transform(animals, get_name));
```

optional

```
auto u = [](std::optional<horse> o){
    return o >>= add_magic;
}
```

optional

```
auto u = [](std::optional<horse> o){
    return o >>= add_magic | add_horn;
}
```

optional

```
auto u = [](std::optional<horse> o){
    return o >>= add_magic >>= add_horn;
}
```

noise

```
auto u = [](std::optional<horse> o){
    return o >>= add_magic >>= add_horn;
}
```

signal

```
auto u = [](std::optional<horse> o){
    return o >>= add_magic >>= add_horn;
}
```

noise

```
auto u = [](std::optional<horse> o){
    return o >>= add_magic >>= add_horn;
}

auto u = identity >>= add_magic >>= add_horn;
```

Terser lambda syntax

```
auto u = [](std::optional<horse> o){
    return o >>= add_magic >>= add_horn;
}

auto u = ~add_magic >>= add_horn;
```

optional

```
auto u = [](std::optional<horse> o){
    return o >>= add_magic >>= add_horn;
}

auto u = ~add_magic >>= add_horn;

u(foo);
```

boost.parameter2

```
constexpr auto scale_image =
            tee(get_image,get_width,get_height) |
            [](Image img, Width w, Height h)->Image
            {
                                    // do the things
            }
```

boost.parameter2

```
constexpr auto scale_image =
            tee(get_image,get_width,get_height) |
            [](Image img, Width w, Height h)->Image
            {
                                    // do the things
            }
```

boost.parameter2

```
auto scale_image

   (Image img, Width w, Height h)->Image
{
                        // do the things
}
```

Simons cat

```
std::optional<image_view> get_cute_cat (image_view img) {
    auto cropped = find_cat(img);
    if (!cropped) {
        return std::nullopt;
    }

    auto with_tie = add_bow_tie(*cropped);
    if (!with_tie) {
        return std::nullopt;
    }

    auto with_sparkles = make_eyes_sparkle(*with_tie);
    if (!with_sparkles) {
        return std::nullopt;
    }

    return add_rainbow(make_smaller(*with_sparkles));
}
```

Simons cat

```
tl::optional<image_view> get_cute_cat (image_view img) {
    return crop_to_cat(img)
            .and_then(add_bow_tie)
            .and_then(make_eyes_sparkle)
            .map(make_smaller)
            .map(add_rainbow);
}
```

Simons cat

```
auto cuteify_cat = crop_to_cat >>=
                   add_bow_tie >>=
                   make_eyes_sparkle >>=
                   make_smaller |
                   add_rainbow;
```

Unwrap a variant?

```
my_variant >>= foo;
```

visit

```
my_variant >>= foo;

std::visit(foo,my_variant);
```

Visit a pointer?

```
void f(int* bar){
    bar >>= baz;
}
```

sum_type

- lightweight variant
- has an explicit nothing state

Composable visitors

```
v >>= partition(pred, foo, bar);
```

Composable visitors

```
v >>= partition(pred, foo, bar);

v >>= subset<T,U> | bar;
```

visit

```
queue.pop() >>= first_match(
        case_(int) => foo,
        case_(char) => use_(a,_1,s) | bar
);
```

visit

```
queue.pop() >>= first_match(
    thing1.dispatch,
    thing2.dispatch,
    thing3.dispatch)
```

state machine

```
variant<S1,S2,S3> sm;
```

state machine

```
variant<S1,S2,S3> sm;
auto transition(S1&,E1&&)->S2;
```

state machine

```
variant<S1,S2,S3> sm;
auto transition(S1&,E1&&)->S2;
auto transition(S1&,E2&&)->S3;
```

```
constexpr auto process_ev = use(~_2, _1) |
    transition >>= assign_to(sm);

process_ev(e,sm);
```

state machine

```
variant<S1,S2,S3> sm;
auto transition(S1&,E1&&)->S2;
auto transition(S1&,E2&&)->S3;
auto transition(S2&,E1&&)->nothing;
```

```
constexpr auto process_ev = use(~_2, _1) |
    transition >>= assign_to(sm);

process_ev(e,sm);
```

state machine

```
variant<S1,S2,S3> sm;
auto transition(S1&,E1&&)->S2;
auto transition(S1&,E2&&)->S3;
auto transition(S2&,E1&&)->nothing;
auto transition(S2&,E2&&)->fragment<S1,nothing>;
```

```
constexpr auto process_ev = use(~_2, _1) |
    transition >>= assign_to(sm);

process_ev(e,sm);
```

state machine

```
variant<S1,S2,S3> sm;
auto transition(S1&,E1&&)->S2;
auto transition(S1&,E2&&)->S3;
auto transition(S2&,E1&&)->nothing;
auto transition(S2&,E2&&)->fragment<S1,nothing>;
auto transition(S3&,E1&&)->nothing;
auto transition(S3&,E2&&)->fragment<S3,S1,nothing>;

constexpr auto process_ev = use(~_2, _1) |
    transition >>= assign_to(sm);

process_ev(e,sm);
```

state machine

```
variant<S1,S2,S3> sm;
auto transition(S1&,E1&&)->S2;
auto transition(S1&,E2&&)->S3;
auto transition(S2&,E1&&)->nothing;
auto transition(S2&,E2&&)->fragment<S1,nothing>;
auto transition(S3&,E1&&)->nothing;
auto transition(S3&,E2&&)->fragment<S3,S1,nothing>;

constexpr auto process_ev = use(~_2, _1) |
    transition >>= assign_to(sm);

process_ev(e,sm);
```

a more powerful state machine

```
//call with event, sm, visitor
constexpr auto process_ev =
    use(~_2, _1,
        _2 | tap<"sm">,
        _3 | tap<"visiter">) |
    tap<"visiter"> >>=
    assign_to(tap<"sm">);
```

a more powerful state machine

```
//call with event, sm, visitor
constexpr auto process_ev =
    use(~_2, _1,
        _2 | tap<"sm">,
        _3 | tap<"visiter">) |
    tap<"visiter"> >>=
    assign_to(tap<"sm">);
```

a more powerful state machine

```
//call with event, sm, visitor
constexpr auto process_ev =
    use(~_2, _1,
        _2 | tap<"sm">,
        _3 | tap<"visiter">) |
    tap<"visiter"> >>=
    assign_to(tap<"sm">);
```

Sub machine

```
auto transition(S2&s,E2 e)->one_of<S1,nothing>{
    return process_ev(e,s,v);
}
```

a more powerful state machine

```
//call with event, sm, visitor
constexpr auto process_ev =
    use(~_2, _1,
        _2 | tap<"sm">,
        _3 | tap<"visiter">) |
    tap<"visiter"> >>= partition(
    tap<"sm">.is_alternative,
    assign_to(tap<"sm">),identity);
```

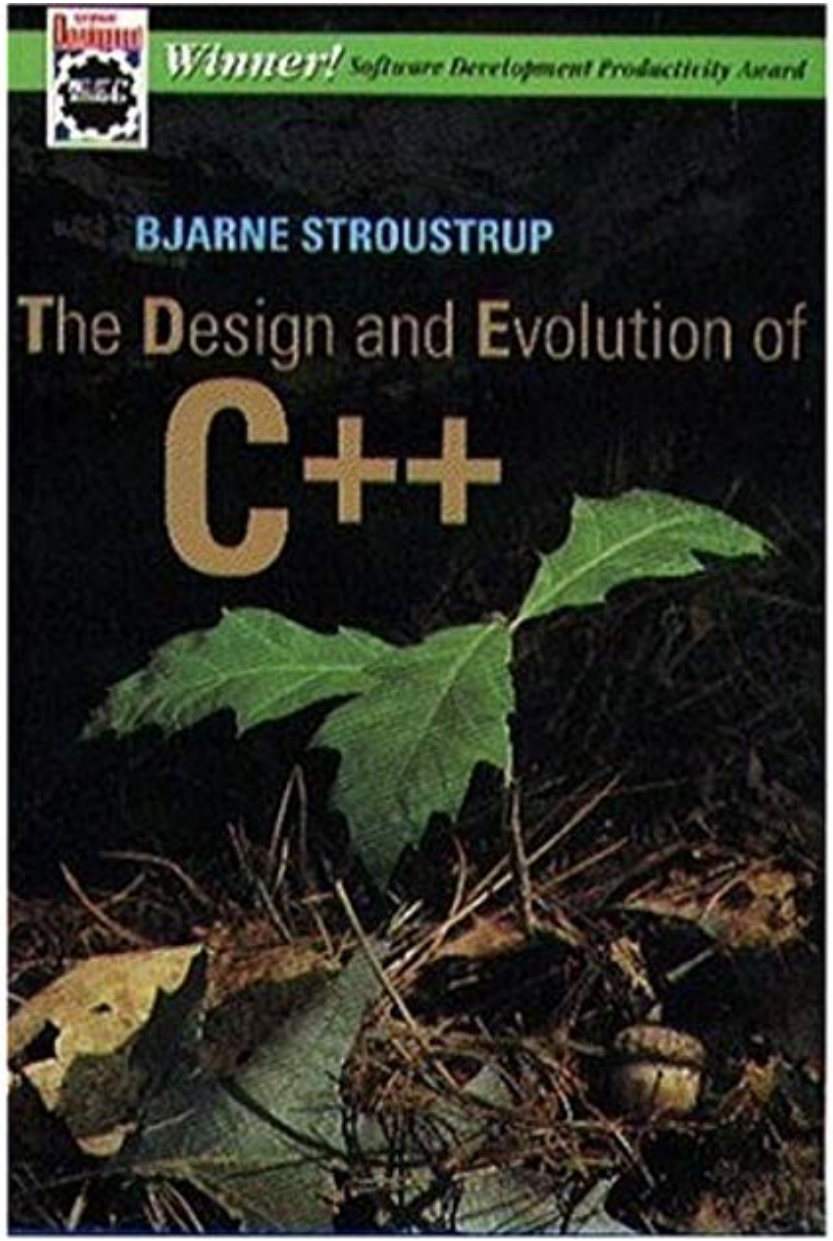
Encapsulated sub-machine

```
template<typename T, typename U>
class reader_sm : sm_stuff<reader_sm>{
    variant<A,B,C> _sm;
    auto operator(A&s,E1&&e)->B;
    auto operator(A&s,E2&&e)->fragment<A,T,U>;
    friend class sm_stuff<reader_sm>;
}
```

```
auto my_uart = mixin(
        /...*/,
        mqtt(my_json) | my_queue
        );


auto handle_event = first_match(
            pwm.dispatch,
            switches.dispatch);

run(handle_event,my_uart,pwm,switches)
```


Winner! Software Development Productivity Award
BJARNE STROUSTRUP
The Design and Evolution of
C++

@odinthenerd

- Github.com
- Twitter.com
- Gmail.com
- Blogspot.com
- LinkedIn.com
- Embo.io