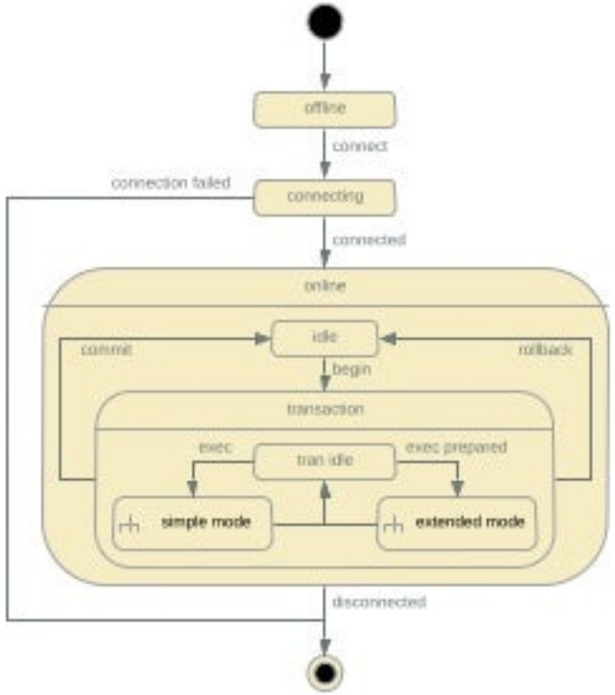
Яндекс Такси

Метапрограммирование: Строим конечный автомат



Яндекс Такси

О чём поговорим

Конечный автомат - что это?

DSL для конечного автомата

Реализация DSL

Где можно применить?

Конечный автомат - что это?

Определение из Википедии

Конечный автомат – абстрактный автомат, число внутренних состояний которого конечно.

https://ru.wikipedia.org/wiki/Конечный_автомат

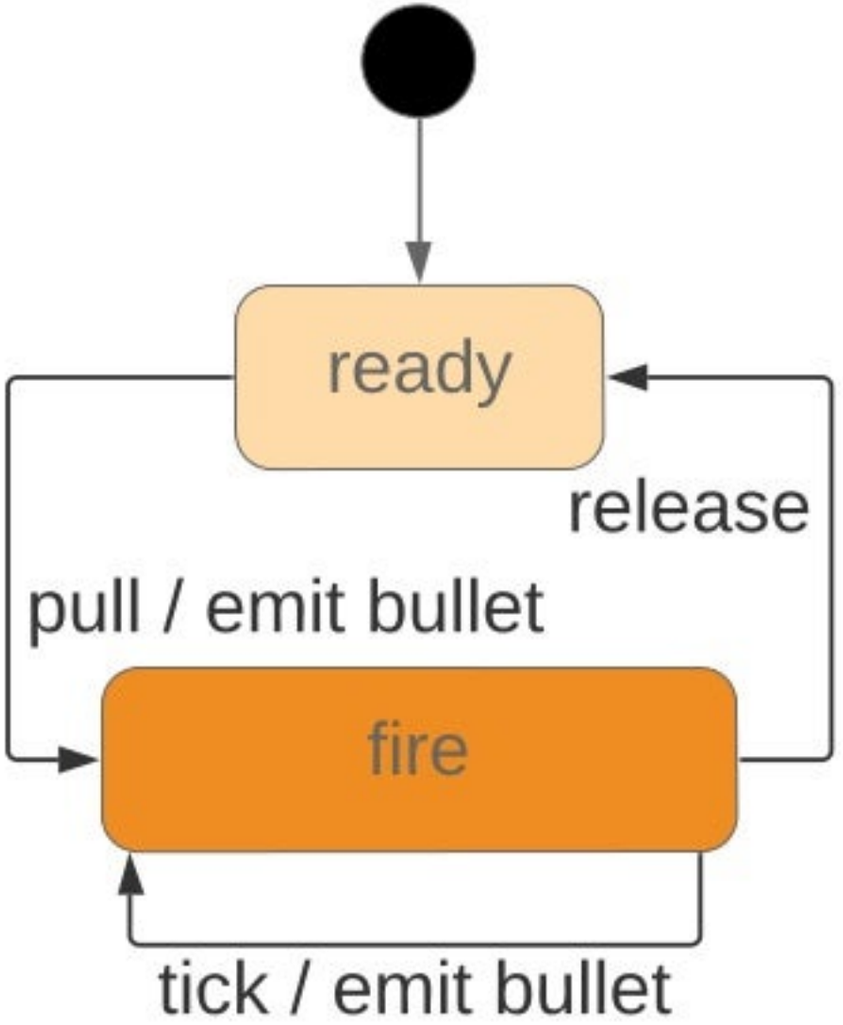
Как выглядит?

$M = (V, Q, q_0, F, \delta)$

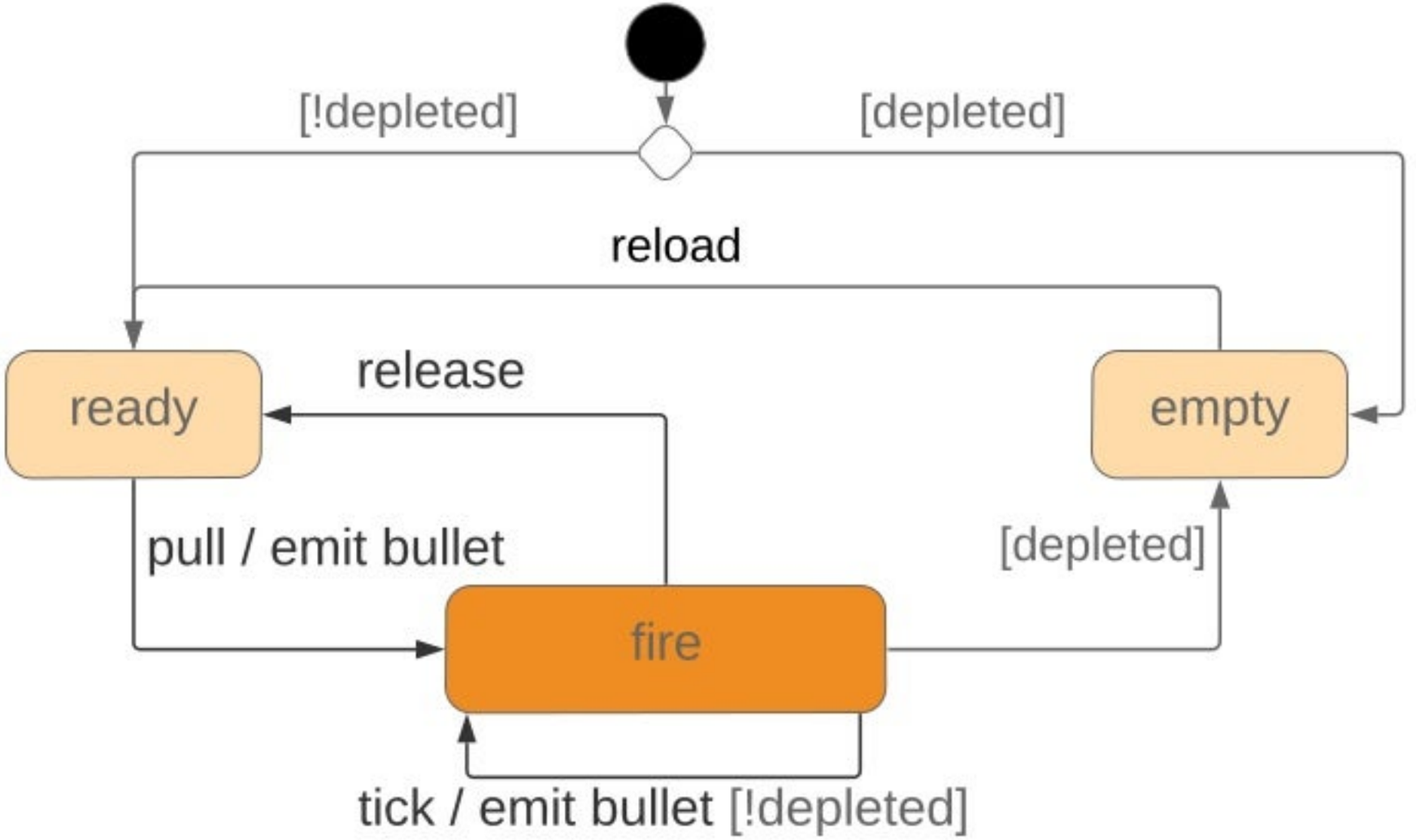
```
/\/\*[\s\S]*?\*\/|([^\\:]|^)\/\/.*$/gm
```

```
statement
    : 'if' paren_expr statement
    | 'if' paren_expr statement 'else' statement
    | 'while' paren_expr statement
    | 'do' statement 'while' paren_expr ';'
    | '{' statement* '}'
    | expr ';'
    | ';'
    ;
```

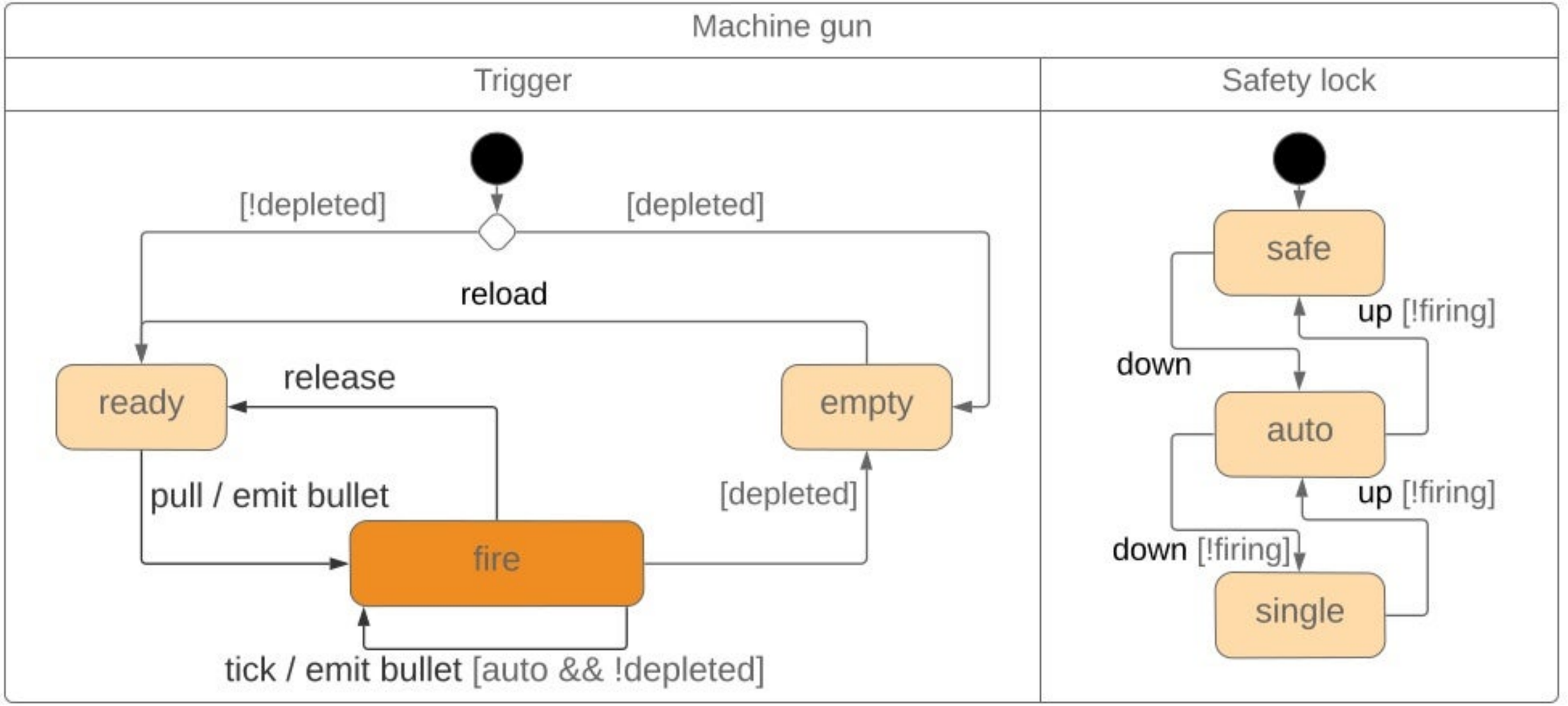
Простейший КА на примере АК



Условные переходы



Ортогональные состояния



Табличное представление

State	Event	Next State	Action	Condition (guard)
		trigger		
initial	-	ready		!depleted
initial	-	empty		depleted
empty	reload	ready	change magazine	
ready	reload	fire	change magazine	
ready	pull the trigger	fire	emit bullet	!safe
fire	release the trigger	ready		
fire	tick	fire	emit bullet	automatic && !depleted
fire	-	empty		depleted
		selector		
safe	lever down	automatic		
automatic	lever up	safe		!firing
automatic	lever down	single		!firing
single	lever up	automatic		!firing

DSL для конечного автомата

Основные сущности

- Машина состояний (state machine)
  - Таблица переходов (transition table)
  - Переход (transition)
  - Событие (event)
  - Состояние (state)
  - Действие (action)
    - при входе в состояние (on entry)
    - при выходе из состояния (on exit)
    - при переходе (transition action)
  - Условие перехода (transition guard)

Основные сущности

Машина состояний (state machine)

Таблица переходов (transition table)

Переход (transition)

Событие (event)

Состояние (state)

Действие (action)

- при входе в состояние (on entry)
- при выходе из состояния (on exit)
- при переходе (transition action)

Условие перехода (transition guard)

КА с ортогональными состояниями

```
struct machine_gun_def : afsm::def::state_machine<machine_gun_def> {
    //@{
    /** @name Sub-machines */
    struct selector;
    struct trigger;

    using orthogonal_regions = type_tuple<trigger, selector>;
    //@}
};
```

Табличное представление

State	Event	Next State	Action	Condition (guard)
		trigger		
initial	-	ready		!depleted
initial	-	empty		depleted
empty	reload	ready	change magazine	
ready	reload	fire	change magazine	
ready	pull the trigger	fire	emit bullet	!safe
fire	release the trigger	ready		
fire	tick	fire	emit bullet	automatic && !depleted
fire	-	empty		depleted
		selector		
safe	lever down	automatic		
automatic	lever up	safe		!firing
automatic	lever down	single		!firing
single	lever up	automatic		!firing

Табличное представление

State	Event	Next State	Action	Condition (guard)
		trigger		
initial	-	ready		!depleted
initial	-	empty		depleted
empty	reload	ready	change magazine	
ready	reload	fire	change magazine	
ready	pull the trigger	fire	emit bullet	!safe
fire	release the trigger	ready		
fire	tick	fire	emit bullet	automatic && !depleted
fire	-	empty		depleted
		selector		
safe	lever down	automatic		
automatic	lever up	safe		!firing
automatic	lever down	single		!firing
single	lever up	automatic		!firing

Машина состояний

```
struct selector : state_machine<selector> {
    /** @name Selector substates */
     struct safe;
     struct single;
     struct automatic;

     using initial_state = safe;

     // Type aliases to shorten the transition table
     using down = events::safety_lever_down;
     using up   = events::safety_lever_up;

    // selector transition table
     using transitions = transition_table<
     /*  State       | Event | Next      | Action          | Guard          */
     tr< safe        , down  , automatic , update_safety , none           >,
     tr< automatic   , up    , safe      , update_safety , not_<firing>   >,
     tr< automatic   , down  , single    , update_safety , not_<firing>   >,
     tr< single      , up    , automatic , update_safety , not_<firing>   >
     >;
};
```

Основные сущности

Машина состояний (state machine)

- ✓ Таблица переходов (transition table)
- ✓ Переход (transition)
- Событие (event)
- Состояние (state)
- Действие (action)
  - при входе в состояние (on entry)
  - при выходе из состояния (on exit)
  - при переходе (transition action)
- Условие перехода (transition guard)

Основные сущности

Машина состояний (state machine)

✓ Таблица переходов (transition table)

✓ Переход (transition)

Событие (event)

Состояние (state)

Действие (action)

при входе в состояние (on entry)

при выходе из состояния (on exit)

при переходе (transition action)

Условие перехода (transition guard)

Описание событий

```
namespace guns::events {

struct trigger_pull {};
struct trigger_release {};

struct reload {};

struct safety_lever_up {};
struct safety_lever_down {};

struct tick {
  std::uint64_t frame = 0;
};

}  // namespace guns::events
```

Описание событий

```
namespace guns::events {

struct trigger_pull {};
struct trigger_release {};

struct reload {};

struct safety_lever_up {};
struct safety_lever_down {};

struct tick {
  std::uint64_t frame = 0;
};

}  // namespace guns::events
```

Основные сущности

Машина состояний (state machine)

- ✓ Таблица переходов (transition table)
- ✓ Переход (transition)
- ✓ Событие (event)
- Состояние (state)
- Действие (action)
  - при входе в состояние (on entry)
  - при выходе из состояния (on exit)
  - при переходе (transition action)
- Условие перехода (transition guard)

Основные сущности

Машина состояний (state machine)

✓ Таблица переходов (transition table)

✓ Переход (transition)

✓ Событие (event)

Состояние (state)

Действие (action)

при входе в состояние (on entry)

при выходе из состояния (on exit)

при переходе (transition action)

Условие перехода (transition guard)

Описание состояний

```
//@{
/** @name Selector substates */
struct safe : state<safe> {
    static constexpr safety_lever lever = safety_lever::safe;
};
struct single : state<single> {
    static constexpr safety_lever lever = safety_lever::single;
};
struct automatic : state<automatic> {
    static constexpr safety_lever lever = safety_lever::automatic;
};
//@}
```

Основные сущности

Машина состояний (state machine)

- ✓ Таблица переходов (transition table)
- ✓ Переход (transition)
- ✓ Событие (event)
- ✓ Состояние (state)
- Действие (action)
  - при входе в состояние (on entry)
  - при выходе из состояния (on exit)
  - при переходе (transition action)
- Условие перехода (transition guard)

Основные сущности

Машина состояний (state machine)

- ✓ Таблица переходов (transition table)
- ✓ Переход (transition)
- ✓ Событие (event)
- ✓ Состояние (state)
- Действие (action)
  - **при входе в состояние (on entry)**
  - **при выходе из состояния (on exit)**
  - при переходе (transition action)
- Условие перехода (transition guard)

Действия при входе/выходе

```
struct empty : state<empty> {
    template <typename FSM>
    void
    on_enter(afsm::none&&, FSM& fsm)
    {
        root_machine(fsm).ammo_depleted();
    }
    template <typename FSM>
    void
    on_exit(events::reload const&, FSM& fsm)
    {
        root_machine(fsm).ammo_replentished();
    }
};
```

Действия при входе/выходе

```
struct empty : state<empty> {
    template <typename FSM>
    void
    on_enter(afsm::none&&, FSM& fsm)
    {
        root_machine(fsm).ammo_depleted();
    }
    template <typename FSM>
    void
    on_exit(events::reload const&, FSM& fsm)
    {
        root_machine(fsm).ammo_replentished();
    }
};
```

Действия при входе/выходе

```
struct empty : state<empty> {
    template <typename FSM>
    void
    on_enter(afsm::none&&, FSM& fsm)
    {
        root_machine(fsm).ammo_depleted();
    }
    template <typename FSM>
    void
    on_exit(events::reload const&, FSM& fsm)
    {
        root_machine(fsm).ammo_replentished();
    }
};
```

Основные сущности

Машина состояний (state machine)

- ✓ Таблица переходов (transition table)
- ✓ Переход (transition)
- ✓ Событие (event)
- ✓ Состояние (state)

Действие (action)

- ✓ при входе в состояние (on entry)
- ✓ при выходе из состояния (on exit)
- при переходе (transition action)

Условие перехода (transition guard)

Основные сущности

Машина состояний (state machine)

- ✓ Таблица переходов (transition table)
- ✓ Переход (transition)
- ✓ Событие (event)
- ✓ Состояние (state)

Действие (action)

- ✓ при входе в состояние (on entry)
- ✓ при выходе из состояния (on exit)
- **при переходе (transition action)**

Условие перехода (transition guard)

Действия при переходе

```
struct selector : state_machine<selector> {
    /** @name Selector substates */
     struct safe;
     struct single;
     struct automatic;

     using initial_state = safe;

     // Type aliases to shorten the transition table
     using down = events::safety_lever_down;
     using up   = events::safety_lever_up;

    // selector transition table
     using transitions = transition_table<
     /*  State       | Event | Next      | Action          | Guard          */
     tr< safe        , down  , automatic , update_safety , none           >,
     tr< automatic   , up    , safe      , update_safety , not_<firing>   >,
     tr< automatic   , down  , single    , update_safety , not_<firing>   >,
     tr< single      , up    , automatic , update_safety , not_<firing>   >
     >;
};
```

Действия при переходе

```
//@{
/** @name Selector substates */
struct safe : state<safe> {
    static constexpr safety_lever lever = safety_lever::safe;
};
struct single : state<single> {
    static constexpr safety_lever lever = safety_lever::single;
};
struct automatic : state<automatic> {
    static constexpr safety_lever lever = safety_lever::automatic;
};
//@}
```

Действия при переходе

```
struct update_safety {
    template <typename Event, typename FSM,
              typename SourceState, typename TargetState>
    void
    operator()(Event const&, FSM& fsm, SourceState&, TargetState&) const
    {
        root_machine(fsm).update_safety(TargetState::lever);
    }
};
```

Действия при переходе

```
struct change_magazine {
    template <typename Event, typename FSM>
    void
    operator()(Event const&, FSM& fsm) const
    {
        root_machine(fsm).reload();
    }
};
```

Действия при переходе

```
struct change_magazine {
    template <typename Event, typename FSM>
    void
    operator()(Event const&, FSM& fsm) const
    {
        root_machine(fsm).reload();
    }
};
```

Основные сущности

Машина состояний (state machine)

- ✓ Таблица переходов (transition table)
- ✓ Переход (transition)
- ✓ Событие (event)
- ✓ Состояние (state)
- ✓ Действие (action)
  - ✓ при входе в состояние (on entry)
  - ✓ при выходе из состояния (on exit)
  - ✓ при переходе (transition action)
- Условие перехода (transition guard)

Основные сущности

Машина состояний (state machine)

- ✓ Таблица переходов (transition table)
- ✓ Переход (transition)
- ✓ Событие (event)
- ✓ Состояние (state)
- ✓ Действие (action)
  - ✓ при входе в состояние (on entry)
  - ✓ при выходе из состояния (on exit)
  - ✓ при переходе (transition action)
- **Условие перехода (transition guard)**

Условия перехода

```
struct selector : state_machine<selector> {
    /** @name Selector substates */
     struct safe;
     struct single;
     struct automatic;

     using initial_state = safe;

     // Type aliases to shorten the transition table
     using down = events::safety_lever_down;
     using up   = events::safety_lever_up;

    // selector transition table
     using transitions = transition_table<
     /*  State       | Event | Next       | Action          | Guard          */
     tr< safe        , down  , automatic , update_safety , none           >,
     tr< automatic   , up    , safe      , update_safety , not_<firing> >,
     tr< automatic   , down  , single    , update_safety , not_<firing> >,
     tr< single      , up    , automatic , update_safety , not_<firing> >
     >;
};
```

Условия перехода

```
struct firing : machine_gun_def::in_state<machine_gun_def::trigger::fire> {};
```

Условия перехода

```
struct firing : machine_gun_def::in_state<machine_gun_def::trigger::fire> {};
```

Условия перехода

```
// trigger transition table
using transitions = transition_table<
/*  State    | Event      | Next       | Action             | Guard                                  */
tr< init     , none       , ready      , none               , not_<depleted>                          >,
tr< init     , none       , empty      , none               , depleted                                >,
tr< empty    , reload     , ready      , change_magazine    , none                                    >,
tr< ready    , pull       , fire       , emit_bullet        , not_<in_state<selector::safe>>          >,
tr< ready    , reload     , ready      , change_magazine    , none                                    >,
tr< fire     , release    , ready      , none               , none                                    >,
tr< fire     , tick       , fire       , emit_bullet        , and_<in_state<selector::automatic>,
                                                                 want_this_frame,
                                                                 not_<depleted>>                       >,
tr< fire     , none       , empty      , none               , depleted                                >
>;
```

Условия перехода

```
struct depleted {
    template <typename FSM, typename State>
    bool
    operator()(FSM const& fsm, State const&) const
    {
        return root_machine(fsm).empty();
    }
};

struct want_this_frame {
    template <typename FSM, typename State>
    bool
    operator()(FSM const&, State const&, events::tick const& t) const
    {
        return t.frame % 10 == 0;
    }
};
```

Основные сущности

Машина состояний (state machine)

- ✓ Таблица переходов (transition table)
- ✓ Переход (transition)
- ✓ Событие (event)
- ✓ Состояние (state)
- ✓ Действие (action)
  - ✓ при входе в состояние (on entry)
  - ✓ при выходе из состояния (on exit)
  - ✓ при переходе (transition action)
- ✓ Условие перехода (transition guard)

Как это использовать в коде?

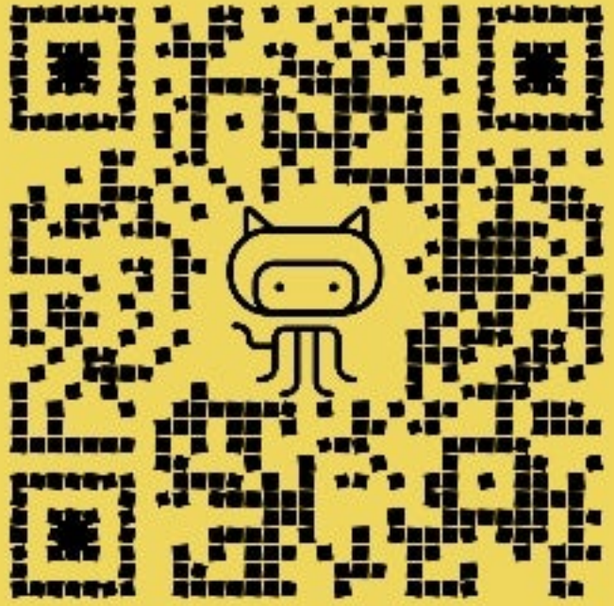
```
namespace guns {
using machine_gun = afsm::state_machine<machine_gun_def>;
}    // namespace guns

int
main(int, char*[])
{
    guns::machine_gun mg;

    mg.process_event(afsm::none{});
    mg.process_event(guns::events::safety_lever_down{});
    mg.process_event(guns::events::reload{});
    mg.process_event(guns::events::trigger_pull{});
    while (!mg.empty()) {
        mg.process_event(guns::events::tick{});
    }
    mg.process_event(guns::events::trigger_release{});

    return 0;
}
```

Реализация DSL



https://github.com/zmij/afsm

Объявление front machine

```
namespace guns {
using machine_gun = afsm::state_machine<machine_gun_def>;
}    // namespace guns

int
main(int, char*[])
{
    guns::machine_gun mg;

    mg.process_event(afsm::none{});
    mg.process_event(guns::events::safety_lever_down{});
    mg.process_event(guns::events::reload{});
    mg.process_event(guns::events::trigger_pull{});
    while (!mg.empty()) {
        mg.process_event(guns::events::tick{});
    }
    mg.process_event(guns::events::trigger_release{});

    return 0;
}
```

Объявление front machine

```
template <typename T, typename Mutex = none, typename Observer = none,
          template <typename> class ObserverWrapper = detail::observer_wrapper>
class state_machine;
```

Transition

```
template <typename SourceState, typename Event, typename TargetState, typename Action = none,
          typename Guard = none>
struct transition {
    using source_state_type = SourceState;
    using target_state_type = TargetState;
    using event_type        = Event;
    using action_type       = Action;
    using guard_type        = Guard;

    using key_type = detail::transition_key<SourceState, Event, Guard>;

    struct value_type {
        using action_type       = transition::action_type;
        using target_state_type = transition::target_state_type;
    };
};
```

Transition table

```
template <typename... T>
struct transition_table {
    static_assert(
        (psst::meta::all_match<traits::is_transition, T...>::type::value),
        "Transition table can contain only transition or internal_transition template "
        "instantiations");
    using transitions     = psst::meta::type_tuple<T...>;
    using inner_states =
        typename psst::meta::unique<typename detail::source_state<T>::type...,
                                    typename detail::target_state<T>::type...>::type;
    using handled_events = typename psst::meta::unique<typename T::event_type...>::type;

    static constexpr std::size_t inner_state_count = inner_states::size;
    static constexpr std::size_t event_count       = handled_events::size;

    static_assert((psst::meta::all_match<traits::is_state, inner_states>::type::value),
                  "State types must derive from afsm::def::state");
    static_assert(transitions::size == transition_count, "Duplicate transition");
};
```

Transition Source State

```
template <typename T>
struct source_state;

template <typename SourceState, typename Event, typename TargetState, typename Action,
          typename Guard>
struct source_state<transition<SourceState, Event, TargetState, Action, Guard>> {
    using type = SourceState;
};

template <typename Event, typename Action, typename Guard>
struct source_state<internal_transition<Event, Action, Guard>> {
    using type = void;
};
```

Маленькие хитрости

```
template <typename StateMachine, typename... Tags>
struct state_machine_def : state_def<StateMachine, Tags…>,
       tags::state_machine {
   template <typename SourceState, typename Event,
             typename TargetState, typename Action = none,
             typename Guard = none>
    using tr = transition<SourceState, Event, TargetState, Action, Guard>;

    template <typename T, typename... TTags>
    using state = doesnt_really_matter;
    template <typename T, typename... TTags>
    using state_machine = doesnt_really_matter;

    using none = afsm::none;
    template <typename Predicate>
    using not_ = psst::meta::not_<Predicate>;
    template <typename... Predicates>
    using and_ = psst::meta::and_<Predicates...>;
    template <typename... Predicates>
    using or_ = psst::meta::or_<Predicates...>;
};
```

none

```
// trigger transition table
using transitions = transition_table<
/*  State    | Event      | Next       | Action              | Guard                                    */
tr< init     , none       , ready      , none                , not_<depleted>                           >,
tr< init     , none       , empty      , none                , depleted                                 >,
tr< empty    , reload     , ready      , change_magazine     , none                                     >,
tr< ready    , pull       , fire       , emit_bullet         , not_<in_state<selector::safe>>           >,
tr< ready    , reload     , ready      , change_magazine     , none                                     >,
tr< fire     , release    , ready      , none                , none                                     >,
tr< fire     , tick       , fire       , emit_bullet         , and_<in_state<selector::automatic>,
                                                                  want_this_frame,
                                                                  not_<depleted>>                       >,
tr< fire     , none       , empty      , none                , depleted                                 >
>;
```

none

```
struct none {};
```

none: action

```
struct none {};

template <typename FSM, typename SourceState, typename TargetState>
struct action_invocation<none, FSM, SourceState, TargetState> {
    template <typename Event>
    void
    operator()(Event&&, FSM&, SourceState&, TargetState&) const
    {}
};
```

none: guard

```
struct none {};

template <typename FSM, typename State, typename Event>
struct guard_check<FSM, State, Event, none> {
    constexpr bool
    operator()(FSM const&, State const&, Event const&) const
    {
        return true;
    }
};
```

none: event

```
struct none {};

void
check_default_transition()
{
    auto const& ttable = transition_table<none>(state_indexes{});
    ttable[current_state()](*this, none{});
}
```

Обработка обычного действия при переходе

```
template <typename Event>
actions::event_process_result
process_transition_event(Event&& event)
{
    auto const& inv_table = transition_table<Event>(state_indexes{});
    return inv_table[current_state()](*this, std::forward<Event>(event));
}
```

Transition table для текущего события

```
template <typename Event, std::size_t... Indexes>
static transition_table_type<Event> const&
transition_table(psst::meta::indexes_tuple<Indexes...> const&)
{
    using event_type = typename std::decay<Event>::type;
    using event_transitions =
        typename psst::meta::find_if<def::handles_event<event_type>::template type,
                                     transitions_tuple>::type;
    static transition_table_type<Event> _table{{
        typename detail::transition_action_selector<fsm_type, this_type,
            typename psst::meta::find_if<
                def::originates_from<
                    typename inner_states_def::template type<Indexes>
                >::template type,
                event_transitions
            >::type
        >::type{}...}};
    return _table;
}
```

Transition table для текущего события

```
template <typename Event, std::size_t... Indexes>
static transition_table_type<Event> const&
transition_table(psst::meta::indexes_tuple<Indexes...> const&)
{
    using event_type = typename std::decay<Event>::type;
    using event_transitions =
        typename psst::meta::find_if<def::handles_event<event_type>::template type,
                                     transitions_tuple>::type;
    static transition_table_type<Event> _table{{
        typename detail::transition_action_selector<fsm_type, this_type,
            typename psst::meta::find_if<
                def::originates_from<
                    typename inner_states_def::template type<Indexes>
                >::template type,
                event_transitions
            >::type
        >::type{}...}};
    return _table;
}
```

Transition table для текущего события

```
template <typename Event, std::size_t... Indexes>
static transition_table_type<Event> const&
transition_table(psst::meta::indexes_tuple<Indexes...> const&)
{
    using event_type = typename std::decay<Event>::type;
    using event_transitions =
        typename psst::meta::find_if<def::handles_event<event_type>::template type,
                                     transitions_tuple>::type;
    static transition_table_type<Event> _table{{
        typename detail::transition_action_selector<fsm_type, this_type,
            typename psst::meta::find_if<
                def::originates_from<
                    typename inner_states_def::template type<Indexes>
                >::template type,
                event_transitions
            >::type
        >::type{}...}};
    return _table;
}
```

meta::find_if

```
namespace detail {

template <template <typename> class Predicate, std::size_t N, typename... T>
struct find_if_impl;

template <template <typename> class Predicate, std::size_t N, typename T, typename... Y>
struct find_if_impl<Predicate, N, T, Y...> {
    using tail = find_if_impl<Predicate, N + 1, Y…>;

    using type    = std::conditional_t<Predicate<T>::value,
                                       combine_t<T, typename tail::type>,
                                       typename tail::type>;
};

}    // namespace detail
template <template <typename> class Predicate, typename... T>
struct find_if : detail::find_if_impl<Predicate, 0, T...> {};
```

meta::find_if

```
namespace detail {

template <template <typename> class Predicate, std::size_t N, typename... T>
struct find_if_impl;

template <template <typename> class Predicate, std::size_t N, typename T, typename... Y>
struct find_if_impl<Predicate, N, T, Y...> {
    using tail = find_if_impl<Predicate, N + 1, Y...>;

    using type     = std::conditional_t<Predicate<T>::value,
                                        combine_t<T, typename tail::type>,
                                        typename tail::type>;
};

}    // namespace detail
template <template <typename> class Predicate, typename... T>
struct find_if : detail::find_if_impl<Predicate, 0, T...> {};
```

meta::find_if

```
namespace detail {

template <template <typename> class Predicate, std::size_t N, typename... T>
struct find_if_impl;

template <template <typename> class Predicate, std::size_t N, typename T, typename... Y>
struct find_if_impl<Predicate, N, T, Y...> {
    using tail = find_if_impl<Predicate, N + 1, Y...>;

    using type    = std::conditional_t<Predicate<T>::value,
                                       combine_t<T, typename tail::type>,
                                       typename tail::type>;
};

}    // namespace detail
template <template <typename> class Predicate, typename... T>
struct find_if : detail::find_if_impl<Predicate, 0, T...> {};
```

meta::find_if

```
namespace detail {

template <template <typename> class Predicate, std::size_t N, typename... T>
struct find_if_impl;

template <template <typename> class Predicate, std::size_t N, typename T, typename... Y>
struct find_if_impl<Predicate, N, T, Y...> {
    using tail = find_if_impl<Predicate, N + 1, Y...>;

    using type    = std::conditional_t<Predicate<T>::value,
                                       combine_t<T, typename tail::type>,
                                       typename tail::type>;
};

}    // namespace detail
template <template <typename> class Predicate, typename... T>
struct find_if : detail::find_if_impl<Predicate, 0, T...> {};
```

meta::find_if

```
namespace detail {

template <template <typename> class Predicate, std::size_t N, typename... T>
struct find_if_impl;

template <template <typename> class Predicate, std::size_t N, typename T, typename... Y>
struct find_if_impl<Predicate, N, T, Y...> {
    using tail = find_if_impl<Predicate, N + 1, Y...>;

    using type     = std::conditional_t<Predicate<T>::value,
                                        combine_t<T, typename tail::type>,
                                        typename tail::type>;
};

}    // namespace detail
template <template <typename> class Predicate, typename... T>
struct find_if : detail::find_if_impl<Predicate, 0, T...> {};
```

Где можно применить?

Где применить?

- Лексические анализаторы
- Сетевые протоколы
- Симуляторы
- Игры

Пример

```
/** @name Transition table for transaction */
using transitions = transition_table<
    /*        Start        Event                          Next                Action                    */
     /*+----------------+------------------------------+-------------------+-------------------------+ */
     tr< starting       , events::ready_for_query      , idle              , transaction_started     >,
     /*+----------------+------------------------------+-------------------+-------------------------+ */
     tr< idle           , events::commit               , exiting           , commit_transaction      >,
     tr< idle           , events::rollback             , exiting           , rollback_transaction    >,
     tr< idle           , error::query_error           , exiting           , rollback_transaction    >,
     tr< idle           , error::client_error          , exiting           , rollback_transaction    >,
     /*+----------------+------------------------------+-------------------+-------------------------+ */
     tr< idle           , events::execute              , simple_query      , none                    >,
     tr< simple_query   , events::ready_for_query      , idle              , none                    >,
     tr< simple_query   , error::query_error           , tran_error        , none                    >,
     tr< simple_query   , error::client_error          , tran_error        , none                    >,
     tr< simple_query   , error::db_error              , tran_error        , none                    >,
     /*+----------------+------------------------------+-------------------+-------------------------+ */
     tr< idle           , events::execute_prepared     , extended_query    , none                    >,
     tr< extended_query , events::ready_for_query      , idle              , none                    >,
     tr< extended_query , error::query_error           , tran_error        , none                    >,
     tr< extended_query , error::client_error          , tran_error        , none                    >,
     tr< extended_query , error::db_error              , tran_error        , none                    >,
     /*+----------------+------------------------------+-------------------+-------------------------+ */
     tr< tran_error     , events::ready_for_query      , exiting           , rollback_transaction    >
>;
```

Спасибо

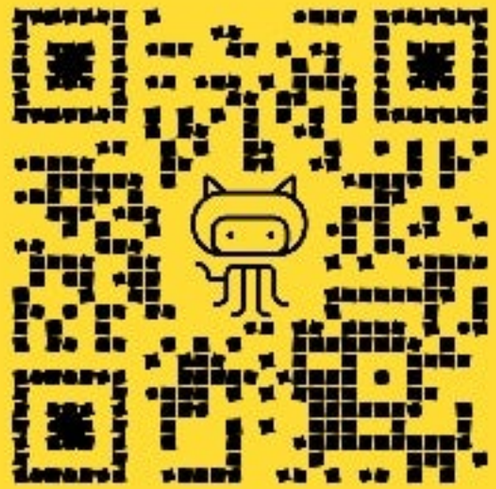
Сергей Федоров

Ведущий разработчик

@zmij_r

ser-fedorov@yandex-team.ru

https://github.com/zmij



Bonus Slides

and_

```
template <typename... Predicate>
struct and_;

template <typename Predicate, typename... Rest>
struct and_<Predicate, Rest...> {
    template <typename... Args>
    bool
    operator()(Args&&... args) const
    {
        return Predicate{}(::std::forward<Args>(args)...)
                && and_<Rest...>{}(::std::forward<Args>(args)...);
    }
};

template <typename Predicate>
struct and_<Predicate> {
    template <typename... Args>
    bool
    operator()(Args&&... args) const
    {
        return Predicate{}(::std::forward<Args>(args)...);
    }
};
```

and_ - C++17

```
template <typename... Predicate>
struct and_ {
    template <typename... Args>
    bool
    operator()(Args&&... args) const
    {
        return (Predicate{}(std::forward<Args>(args)...) && ...);
    }
};
```