Future work, Kvasir::Msm

a look at boost MSM

```
struct transition_table : mpl::vector<
      //    Start     Event         Next      Action                 Guard
      //    +---------+-------------+---------+----------------------
      _row < Stopped , play        , Playing                       >,
      _row < Stopped , open_close  , Open                           >,
      _row < Stopped , stop        , Stopped                       >,
      //    +---------+-------------+---------+----------------------
      _row < Open    , open_close  , Empty                          >,
      //    +---------+-------------+---------+----------------------
      _row < Empty   , open_close  , Open                           >,
      _row < Empty   , cd_detected , Stopped                       >,
      //    +---------+-------------+---------+----------------------
      _row < Playing , stop        , Stopped                       >,
      _row < Playing , pause       , Paused                       >,
      _row < Playing , open_close  , Open                          >,
      //    +---------+-------------+---------+----------------------
      _row < Paused  , end_pause   , Playing                       >,
      _row < Paused  , stop        , Stopped                       >,
      _row < Paused  , open_close  , Open                          >
      //    +---------+-------------+---------+----------------------
> {};
```

```
struct Paused : public msm::front::state<>
{
	template <class Event,class FSM>
	void on_entry(Event const&,FSM& ) {
		/*std::cout << "entering: Paused" << std::endl;*/
	}
	template <class Event,class FSM>
	void on_exit(Event const&,FSM& ) {
		/*std::cout << "leaving: Paused" << std::endl;*/
	}
};
```

```
//states
const auto stopped = []{};
const auto open = []{};
const auto empty = []{};
const auto paused = []{};
//events
const auto play = []{};
const auto openClose = []{};
const auto stop = []{};
const auto cdDetected = []{};
const auto pause = []{};
```

```
using namespace SM = Kvasir::Msm;
auto sm = SM::make(
            SM::initial(stopped),
            SM::transition(stopped, play, playing),
            SM::transition(stopped, openClose, open),
            SM::transition(stopped, stop, stopped),
            SM::transition(open, openClose, empty),
            SM::transition(empty, openClose, open),
            SM::transition(empty, cdDetected, stopped),
            SM::transition(playing, stop, stopped),
            SM::transition(playing, pause, paused),
            SM::transition(playing, openClose, open),
            SM::transition(paused, pause, playing),
            SM::transition(paused, stop, stopped),
            SM::transition(paused, openClose, open),
            SM::onEntry(stopped,[]{/*std::cout<<"entering: Stopped" << std::endl;*/}),
            SM::onExit(stopped,[]{/*std::cout<<"exiting: Stopped" << std::endl;*/})
      );

postEvent(sm,play);
postEvent(sm,stop);
bool b = isInState(sm, stopped);
```

```
struct MyState{
	int i_;
	int j_;
};

constexpr MyState myState;

struct MyEvent{
	int i_;
};

constexpr MyEvent myEvent;

onEntry(myState, [](auto context, MyEvent event){
	getContext(myState,context).i_ = event.i_;
})

transition(oldState, myEvent, myState, [](auto context, MyEvent ev){
		/*executed in old state*/
	},
	[](auto context, MyEvent ev){
		return ev.i_ == 42;
	})
```

```
void HD44780::toggleEnable(){
    wait(ENABLE_WAIT);
    *_E = 1;
    wait(ENABLE_WAIT);
    *_E = 0;
    wait(ENABLE_WAIT);
}

void HD44780::reset(){
    _data->output();

    *_E = 0;
    *_RS = 0;
    *_RW = 0;
    *_data = 0x03;

    toggleEnable();
    wait(RESET_WAIT);

    toggleEnable();
    wait(RESET_WAIT);

    toggleEnable();
    wait(RESET_WAIT);
}
```

```
const auto enableWait = [](auto context){ get(context, root).setTimer(1000); };
const auto resetWait = [](auto context){ get(context, root).setTimer(2000); };

const auto isTimerGuard = [](auto context, EvTimer){ return true; };
constexpr auto sp = Register::sequencePoint;

const auto toggleEnable = SM::chain(
             enableWait,  isTimerGuard,
             set(enablePin),
             enableWait,  isTimerGuard,
             clear(enablePin),
             enableWait, isTimerGuard);

const auto reset = SM::chain(
             makeOutput(dataPort),
             clear(enable,rs,rw),
             sp,
             write(dataPort,Register::value<0x03>()),
             call(toggleEnable),
             resetWait,isTimerGuard,
             call(toggleEnable),
             resetWait,isTimerGuard,
             call(toggleEnable),
             resetWait,isTimerGuard
             );
```

```
auto lcdSm = SM::make(
             super(root),
             initial(idle,call(reset)/*...*/)
             /*...*/);
```

```
if(something){
	/*...*/
	wait();
}
else if(otherthing){
	wait();
}
else {}

const auto c = SM::chain(
		cIf(
				[](auto context,Ev ev){ return ev.something; },
				enableWait,  isTimerGuard),
		cElseIf(
				[](auto context,Ev ev){ return ev.otherthing; },
				enableWait,  isTimerGuard)
				),
		cElse(),
		);
```

Questions?