Maszyna stanów skończonych

Maszyna stanów skończonych (*Finite State Machine*, FSM) to model obliczeniowy opisujący system, który przechodzi między ograniczoną liczbą stanów w odpowiedzi na zdarzenia. Każdy stan reprezentuje unikalny etap lub konfigurację systemu, a przejścia między stanami są zdefiniowane przez reguły zależne od wejściowych zdarzeń.

W C# definiowanie **maszyny stanów skończonych** (*Finite State Machine*, FSM) można uprościć dzięki bibliotece **Stateless**, która pozwala na tworzenie elastycznych i czytelnych maszyn stanów, gdzie logika przejść i akcji jest zdefiniowana w sposób deklaratywny, bez konieczności przechowywania stanu w zmiennych instancji.

📘 Teoretyczna definicja:

Automat skończony to piątka:

```
A = (Q, Σ, δ, q₀, F)
```

Symbol	Znaczenie	Odpowiednik w Stateless
`Q`	Zbiór **stanów**	Enum `State` – np. `Draft`, `Published`, `Archived`
`Σ`	**Alfabet wejściowy** – zbiór symboli	Enum `Trigger` – np. `Publish`, `Archive`, `Restore`
`δ`	**Funkcja przejścia**: `Q × Σ → Q`	`machine.Configure(...).Permit(...)`
`q₀`	**Stan początkowy**	Przekazany w konstruktorze: `new StateMachine<State, Trigger>(State.Draft)`
`F`	**Zbiór stanów akceptujących**	Musisz zdefiniować to samodzielnie (np. `State.Published` jako "akceptujący")

📌 Czyli:

- Jeśli alfabet wejściowy to `{0,1}` → przejścia mogą być tylko `0` lub `1`
- Jeśli alfabet wejściowy to `{a,b}` → przejścia to `a` i `b`
- W przypadku np. Stateless w C# alfabetem są `enum Trigger`, np. `Publish`, `Archive`

🔧 Przykład

Dla alfabetu `{a, b}`:

```
[q0] --a--> [q1]
[q1] --b--> [q2]
```

Dla alfabetu `{Publish, Archive}`:

```
[Draft] --Publish--> [Published]
[Published] --Archive--> [Archived]
```

📘 Definicja maszyny stanów (na podstawie Stateless)

Maszyna stanów:

- ✔ Ma skończony zbiór stanów (`State`) – np. `Draft` , `Published` , `Archived`

Przykład:

```
enum ArticleState { Draft, Published, Archived }
```

- ✔ Reaguje na zdarzenia (`Trigger`) – np. `Publish` , `Archive`
  Przykład:

```
enum ArticleTrigger { Publish, Archive }
```

- ✔ Może definiować akcje wejścia (`OnEntry`) i wyjścia (`OnExit`) z danego stanu
  Przykład:

```
machine.Configure(ArticleState.Published)
       .OnEntry(() => Console.WriteLine("Opublikowano artykuł"));
```

- ✔ Może mieć warunki przejścia (`PermitIf`) – tzw. strażników
  Przykład:

```
machine.Configure(ArticleState.Draft)
       .PermitIf(ArticleTrigger.Publish, ArticleState.Published, () =>
IsReadyToPublish());
```

Przykład z uzyciem Stateless

1. **Instalacja biblioteki Stateless**

Dodaj pakiet Stateless przez NuGet:

`dotnet add package Stateless`

2. Kod implementacji FSM
   Poniższy przykład przedstawia prostą maszynę stanów dla publikacji artykułu.

```
enum State { Draft, Published, Archived }
enum Trigger { Publish, Archive, Restore }

var machine = new StateMachine<State, Trigger>(State.Draft);

machine.Configure(State.Draft)
    .Permit(Trigger.Publish, State.Published);

machine.Configure(State.Published)
    .Permit(Trigger.Archive, State.Archived);

machine.Configure(State.Archived)
    .Permit(Trigger.Restore, State.Draft);

// Generowanie PlantUML
string diagram = UmlDotGraph.Format(machine.GetInfo());
Console.WriteLine(diagram);

machine.Fire(Trigger.Publish);
```

🔍 Odpowiedniki:

Element	Wartość w kodzie
`Q`	`{ Draft, Published, Archived }`
`Σ`	`{ Publish, Archive, Restore }`
`δ`	Zdefiniowane przejścia `.Permit(...)`
q_0	`State.Draft`
`F`	np. `{ State.Published }` – możesz uznać, że to stan "zaakceptowany"

📌 Uwaga:

```
if (machine.State == State.Published)
    Console.WriteLine("Stan akceptujący");
```

✅ Przykład prostego cyklu życia zamówienia:

```
New → Confirmed → Shipped → Delivered
                      ↘
```

```
                    Cancelled
```

Możliwe stany (Q):

```
enum OrderState
{
    New,
    Confirmed,
    Shipped,
    Delivered,
    Cancelled
}
```

🔍 Co może być stanem akceptującym (F)?

To zależy od reguł Twojej domeny. Oto typowe opcje:

Stan	Czy akceptujący?	Uzasadnienie
`Delivered`	✅ Tak	Zamówienie zakończone sukcesem, wszystko się powiodło
`Cancelled`	✅ Tak (czasami)	Zamówienie świadomie anulowane – też jest zakończone
`Shipped`	❌ Nie	W trakcie realizacji – nie zakończone
`New`	❌ Nie	Dopiero utworzone – nic jeszcze się nie wydarzyło

🧠 Praktyczna definicja stanu akceptującego:

Stan, w którym proces może się zakończyć, a dalsze akcje nie są wymagane.

📦 W kodzie:

```
var acceptingStates = new[] { OrderState.Delivered, OrderState.Cancelled
};

if (acceptingStates.Contains(orderStateMachine.State))
{
    Console.WriteLine("Zamówienie zakończone.");
}
```

Graf

Graf moze być wyeksportowany na podstawie maszyny stanów.

DOT graph

```
string graph = UmlDotGraph.Format(phoneCall.GetInfo());
```

Format ten może być renderowany przez strony takie jak
graphviz.org http://www.webgraphviz.com lub viz.js

Mermaid graph

```
string graph = MermaidGraph.Format(machine.GetInfo());
```

Format ten może być renderowany przez GitHub markdown lub silnik aplikacji **Obsidian**.

✅ Zalety Stateless

- Czytelna i deklaratywna konfiguracja
- Obsługa warunków (np. `PermitIf`)
- Obsługa wejścia/wyjścia (`OnEntry`, `OnExit`)
- Wsparcie dla hierarchii stanów i stanów podległych

Biblioteki dla innych języków programowania

- C++ Automaton

Przykład:
https://github.com/tinkerspy/Automaton/blob/master/examples/blink/blink.ino