Тут має бути титулка

Анотація

У роботі досліджені та реалізовані деякі алгоритми над скінченними автоматами. За допомогою детермінованих скінченних автоматів розроблений функціонал генерації ланцюжків, що належать мові автомата, та перевірка належності ланцюжків мові автомата. Запрограмовані важливі операції над скінченними автоматами: об’єднання, конкатенація, замикання та процес мінімізації.

Зміст

Вступ

Теорія скінчених автоматів – один із найперших і найважливіших кроків у вивченні теорії формальних мов. Автомати дозволяють описати лексеми формальних мов, їх синтаксис, семантику [1, с .4]. У свою чергу, формальні мови застосовуються у створенні мов програмування, машинному перекладі тощо.

Робота виконана в системі комп'ютерної алгебри *Wolfram Mathematica 10* і складається з семи розділів в яких теоретичні відомості про операції над скінченними автоматами поєднані із програмними реалізаціями на мові *Wolfram Language* та прикладами їх застосування.

Wolfram Language – мультипарадигмальна мова високого рівня, що є основною мовою системи *Mathematica*. Вона спроектована і розробляється як максимально універсальна мова з акцетом на символьні обчислення, функціональне програмування, логічне програмування.

Мова охоплює багато, часто вузько спеціалізованих, сфер знань. Наприклад, вона має вбудовані функції для роботи з машинами Тюрінга, створення та обробки зображень та звуку, аналізу 3D-моделей, розв'язку диференціальних рівнянь.

Усі елементи програмування мовою *Wolfram,* що використані в даній роботі, задокументовані в довідці системи *Mathematica 10*.

Шість розділів включають в себе реалізації таких операцій: графічне зображення діаграми переходів скінченних автоматів, генерація ланцюжків, що допускають детерміновані скінченні автомати, їх мінімізація, булеві операції над ними та побудова так званого деревоподібного скінченного автомата – автомата, що описує скінченну регулярну мову. В останньому розділі наведені два приклади програм з графічним інтерфейсом, що побудовані на основі вбудованої функції **`Manipulate`** і написаного функціоналу по операціях над автоматами.

1 Означення та програмні представлення скінченних автоматів

1.1 Означення та представлення

Скінченний автомат — це п'ятірка $(Q, \Sigma, \delta, s, F)$, де:

- Q – скінченна множина ***станів***;
- Σ – скінченна множина ***вхідних символів***;
- δ – ***функція переходу***, що визначена як $\delta : Q \times \Sigma \mapsto Q$ для ***детермінованого*** автомата (ДСА), і як $\delta : Q \times \Sigma \mapsto 2^Q$ для ***недетермінованого*** (НСА);
- $s \in Q$ – ***початковий стан***;
- $F \subset Q$ – ***множина заключних станів***.

В даній роботі скінченний автомат представляється як список $\{Q, \Sigma, \delta, s, F\}$, де:

- Q та Σ – списки;
- δ – двовимірний масив (матриця), такий, що $\delta[\![i, j]\!]$ – це стан (у випадку ДСА) або список станів (у випадку НСА), в які автомат перейде з $Q[\![i]\!]$ при вхідному символі $\Sigma[\![i]\!]$;
- s та F – індекси відповідно початкового та заключних станів.

НСА, в алфавіті якого міститься ε (пуста стрічка), називатимемо ***НСА з ε-переходами*** (εНСА).

Робота скінченного автомату полягає у виконанні послідовності ***тактів***. Такт визначається поточним станом автомата і поточним символом, що зчитується із ***вхідного ланцюжка*** ω. Вважається, що εНСА може переходити по символах ε між тактами.

$(q, \omega) \in Q \times \Sigma^*$ – ***конфігурація*** автомата.

Конфігурація (s, ω) називається ***початковою***, а (q, ε), $q \in F$ – ***заключною***.

Детермінований (недетермінований) автомат *переходить* з конфігурації $(q, a\omega)$, $a \in \Sigma$, $\omega \in \Sigma^*$ в конфігурацію (p, ω), якщо $p = (\in)\, \delta(q, a)$. Позначення: $(q, a\omega) \longmapsto (p, \omega)$

Робота скінченного автомата – послідовність конфігурацій. Нехай $K_0, K_1, \ldots, K_n$ – деяка послідовність конфігурації автомата. Тоді $K_0 \mapsto K_1 \bigwedge K_0 \mapsto K_1 \bigwedge \ldots \bigwedge K_{n-1} \mapsto K_n$ коротко позначається як $K_0 \overset{*}{\mapsto} K_n$.

Автомат ***допускає*** ланцюжок ω, якщо $(q_0, \omega) \overset{*}{\mapsto} (q, \varepsilon)$, $q \in F$. [1, с.20]

Опишемо функції-індикатори допустимості ланцюжка скінченним автоматом.

```
DFAAcceptedQ[ω_, {_, Σ_, δ_, s_, F_}] :=
 MemberQ[F, Fold[δ[[##]] &, s, Characters@ω /. Thread[Σ → Range@Length@Σ]]]
NFAAcceptedQ[ω_, {_, Σ_, δ_, s_, F_}] := AnyTrue[Fold[Union @@ δ[[##]] &,
    {s}, Characters@ω /. Thread[Σ → Range@Length@Σ]], MemberQ[F, #] &]

(* Приклад 1. Перевіримо чи задні ланцюжки
 мають парну кількість як літер "a" так і "b". *)
Grid@Transpose[
  {InputForm@#, DFAAcceptedQ[#, {, {"a", "b"}, {{4, 2}, {3, 1},
        {2, 4}, {1, 3}}, 1, {1}}]} & /@ {"", "baba", "bab", "aba"}]
 ""   "baba"  "bab"  "aba"
True   True   False  False

(* Приклад 2. Перевіримо чи задані ланцюжки
 мають в собі підстрічку "abba" або "baab". *)
Grid@Transpose[
  {InputForm@#, NFAAcceptedQ[#, {, {"a", "b"}, {{{1, 2}, {1, 5}}, {{}, {3}},
        {{}, {4}}, {{8}, {}}, {{6}, {}}, {{7}, {}}, {{}, {8}}, {{8}, {8}}},
       1, {8}}]} & /@ {"aabba", "baaba", "abbba", "ababa"}]
"aabba"  "baaba"  "abbba"  "ababa"
 True     True     False    False
```

Використовуючи функцію **FoldList** замість **Fold** маємо можливість виводити послідовність конфігурацій автоматів.

```
DFAConfigurations[ω_, {Q_, Σ_, δ_, s_, _}] := Transpose@
  {Q[[FoldList[δ[[##]] &, s, Characters@ω /. Thread[Σ → Range@Length@Σ]]]],
   Array[StringDrop[ω, #] &, StringLength@ω, 0] // Append["ε"]}
NFAConfigurations[ω_, {Q_, Σ_, δ_, s_, _}] := Thread /@ Transpose@
   {Q[[#]] & /@ FoldList[Union @@ δ[[##]] &,
      {s}, Characters@ω /. Thread[Σ → Range@Length@Σ]],
    Array[StringDrop[ω, #] &, StringLength@ω, 0] // Append["ε"]}

(* Приклад 3. *)
NFAConfigurations["abba", {CharacterRange["1", "8"],
  {"a", "b"}, {{{1, 2}, {1, 5}}, {{}, {3}}, {{}, {4}}, {{8}, {}},
   {{6}, {}}, {{7}, {}}, {{}, {8}}, {{8}, {8}}}, 1, {8}}]
{{{1, abba}}, {{1, bba}, {2, bba}}, {{1, ba}, {3, ba}, {5, ba}},
 {{1, a}, {4, a}, {5, a}}, {{1, ε}, {2, ε}, {6, ε}, {8, ε}}}
```

1.2 Тестові приклади автоматів

Надалі приклади роботи програм будемо демонструвати на наперед визначених автоматах [3, I, 2]. Створимо асоціативний масив для зберігання прикладів ДСА:

```
DFA = <|
   "OddA"(* допускає ланцюжки із непарною кількістю символів "a" *) →
    {{0, 1}, {"a", "b"}, {{2, 1}, {1, 2}}, 1, {2}},
   "AA"(* допускає ланцюжки із підстрічкою "aa" *) →
    {{0, 1, 2}, {"a", "b"}, {{2, 1}, {3, 1}, {3, 3}}, 1, {3}},
   "SecondB"(* допускає ланцюжки, другий символ яких – "b" *) →
    {Range[0, 3], {"a", "b"}, {{2, 2}, {4, 3}, {3, 3}, {4, 4}}, 1, {3}},
   "OneB"(* допускає ланцюжки лише з одним символом "b" *) →
    {{1, 2, 3, 4, 5, 6}, {"a", "b"},
     {{2, 3}, {1, 4}, {5, 6}, {5, 6}, {5, 6}, {6, 6}}, 1, {3, 4, 5}},
   "DifAB"(* допускає ланцюжки, що не мають поруч
     однакових символів *) → {{0, 1, 2, 3}, {"a", "b"},
     {{2, 3}, {4, 3}, {2, 4}, {4, 4}}, 1, {1, 2, 3}},
   "EvenEven"(* допускає ланцюжки, що мають парну
     кількість символів і "a", і "b" *) →
    {{0, 1, 2, 3}, {"a", "b"}, {{4, 2}, {3, 1}, {2, 4}, {1, 3}}, 1, {1}},
   "Mod5"(* допускає бінарні ланцюжки, що представляють собою числа,
    остача при діленні яких на 5 рівна 1 *) → {{0, 1, 2, 3, 4},
     {0, 1}, {{1, 2}, {3, 4}, {5, 1}, {2, 3}, {4, 5}}, 1, {2}},
   "EvenEvenAny"(* те саме, що й EvenEven з довільною
     кількістю символів "c" *) → {{0, 1, 2, 3}, {"a", "b", "c"},
     {{4, 2, 1}, {3, 1, 2}, {2, 4, 3}, {1, 3, 4}}, 1, {1}},
   "DifABC"(* те саме, що й Dif2 з додатковим символом "c" *) →
    {{0, 1, 2, 3, 4}, {"a", "b", "c"},
     {{2, 3, 4}, {5, 3, 4}, {2, 5, 4}, {2, 3, 5}, {5, 5, 5}}, 1, {1, 2, 3, 4}}
   |>;
```

Автомат **ModMxN** є узагальненням **Mod5**. Він допускає ті ланцюжки, що являють собою числа в базі M, остача при діленні яких на N дорівнює x. Використовуючи програмну нотацію даний автомат описується так:

(DFAAcceptedQ[ω, ModMxN[m, x, n]] == True) ⇔

(Mod[FromDigits[ω, m], n] == x)

```
ModMxN[m_, x_, n_] :=
 With[{Q = Range[n]}, {Q, Join[Range[0, 9], CharacterRange["a", "z"]][[;; m]],
   Partition[Flatten@ConstantArray[Q, m], m], 1, {1 + x}}]
```

2 Графічне зображення автоматів

Графічне зображення автоматів базуватимемо на вбудованих функціях промальовування графів **GraphPlot**, **LayeredGraphPlot**. Для цього потрібно підготувати автомат відповідно до синтаксису даних функцій.

2.1 Перетворення матриці переходів у список ребер графа

Для побудови ребер графа ДСА потрібно створити список всеможливих списків-пар $\{p \to q, a\}$, $(p, q) \in Q^2$, $a \in \Sigma$. Для ефективності ми залишимо індекси на p, q замість самих p, q так, як представлена матриця переходів, а також індекс на a замість самого a. Використовуючи той факт, що функція переходів представлена як матриця, застосуємо дві операції внутрішнього (матричного) добутку. У першому добутку замість операції додавання використовується функція побудови списку (**List**), а замість операції множення – функція побудови правила (**Rule**). Аналогічно використовується другий добуток, в якому за допомогою функцій побудови списків отримуємо матрицю списків-ребер, яку залишається лише "вирівняти" (**Flatten**) в одновимірний масив ребер.

Для побудови ребер графа НСА використовується аналогічна процедура матричних множень використовуючи функцію "пронизання списків" (**Thread**).

```
DFAδEdges@δ_ := With[{rl = Range /@ Dimensions@δ}, Flatten[
    Inner[Reverse@*List, rl⟦2⟧, Inner[Rule, rl⟦1⟧, δ, List], List], 1]]
NFAδEdges@δ_ := With[{rl = Range /@ Dimensions@δ},
  Flatten[Inner[Thread@*Reverse@*List, rl⟦2⟧,
    Inner[Thread@*Rule, rl⟦1⟧, δ, Flatten@*List], List], 1]]
```

```
(* Приклад 1. Матиця переходів і список ребер автомата AA. *)
MapAt[MatrixForm, 1]@{#, DFAδEdges@#} &@DFA["AA"]⟦3⟧
```

$$\left\{\begin{pmatrix} 2 & 1 \\ 3 & 1 \\ 3 & 3 \end{pmatrix}, \{\{1 \to 2, 1\}, \{1 \to 1, 2\}, \{2 \to 3, 1\}, \{2 \to 1, 2\}, \{3 \to 3, 1\}, \{3 \to 3, 2\}\}\right\}$$

```
(* Приклад 2. Матриця переходів і список ребер деякого εНСА. *)
MapAt[MatrixForm, 1]@{#, NFAδEdges@#} &@
 {{{2}, {}, {1}}, {{}, {1}, {}}, {{}, {2, 3}, {2}}}
```

$$\left\{\begin{pmatrix} \{2\} & \{\} & \{1\} \\ \{\} & \{1\} & \{\} \\ \{\} & \{2, 3\} & \{2\} \end{pmatrix},\right.$$

$$\left.\{\{1 \to 2, 1\}, \{2 \to 1, 2\}, \{3 \to 2, 2\}, \{3 \to 3, 2\}, \{1 \to 1, 3\}, \{3 \to 2, 3\}\}\right\}$$

Відтак в структурі графа стани залишаться числами (індексами Q), а надписи до станів і ребер зображатимуться за допомогою функцій, описаних в наступних пунктах.

Доцільно провести ще один крок у побудові ребер. Для кращого візуального сприйняття об'єднуватимемо ребра $\{p \to q, a\}$, $\{p \to q, b\}$ (ребра між однаковими станами з однаковими напрямками) в одне ребро $\{p \to q, \{a, b\}\}$. Для цього опишемо функцію, що обробить вже готовий результат **DFAδEdges**:

```
DFACollapseEdges@e_ := e //.
  {h___, {x_, l1_}, u___, {x_, l2_}, t___} :> {h, {x, Flatten@{l1, l2}}, u, t}
```

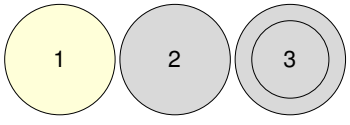
Зауваження. Алгоритм, використаний в **DFACollapseEdges**, дозволяє продемонструвати концепцію "програмування правилами заміни" проте не є найоптимальнішим, але втрата в швидкості виконання коду для прикладів незначуща, а зображення великих автоматів є не практичним в рамках роботи.

2.2 Зображення станів (вершин графа)

Для побудови зображення станів досить зручно використати можливості створення функцій з аргументами-опціями, що дозволяють користувачеві налаштовувти вигляд зображення.

Наступна функція створює список графічних примітивів, що міститимуться в кінцевому результаті – об'єкті векторного зображення **Graphics**. Опції функції включають в себе кольори початкового та кінцевих станів, розмір, функцію що ставитеме у відповідність число (так, як зберігатимуться стани) надпису (так, сприйматиме користувач). Повна інформація промальовки стану – це стан, що промальовується, початковий стан, заключні стани і позиція в декартових координатах.

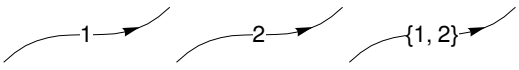
```
Options@DFARenderState =
  {StartColor → LightYellow, FinalColor → LightGray, VertexSize → .1,
   VertexLabeling → Automatic, VertexLabelStyle → {Black, 12}};
DFARenderState[q_, s_, F_, p_, OptionsPattern[]] :=
 {If[q === s, OptionValue@StartColor, OptionValue@FinalColor],
  EdgeForm@Black, Disk[p, OptionValue@VertexSize],
  If[MemberQ[F, q], Disk[p, .7 OptionValue@VertexSize]],
  Text[Style[Replace[OptionValue@VertexLabeling, Automatic → Identity]@q,
    Sequence @@ OptionValue@VertexLabelStyle], p]}

(* Приклад 1. Початковий, звичайний та заключний стани. *)
Row[Graphics[DFARenderState[#, 1, {3}, {0, 0}], ImageSize → 60] & /@ Range@3]
```



2.3 Зображення переходів (ребер графа)

Побудова зображення переходів, як і станів, полягає у створені списку графічних примітивів. В нашому випадку цими примітивами є B-сплайнова крива зі стрілкою, і сам надпис символів переходу. Відносні позиції стрілок і надписів можна налаштовувати, як і розмір стрілок, стиль надпису і функцію-відповідність число-надпис.

```
Options@DFARenderTransition =
  {ArrowSize → Automatic, ArrowPosition → .8, EdgeLabeling → Automatic,
   EdgeLabelPosition → .5, EdgeLabelStyle → {Black, 12}};
DFARenderTransition[t_, p_, OptionsPattern[]] :=
 {Black,
  Arrowheads@{{OptionValue@ArrowSize, OptionValue@ArrowPosition}},
  Arrow@BSplineCurve[p, SplineDegree → Length@p - 1],
  Text[Style[Replace[OptionValue@EdgeLabeling, Automatic → Identity]@t,
    Sequence @@ OptionValue@EdgeLabelStyle],
   BSplineFunction[p, SplineDegree → Length@p - 1]@
    OptionValue@EdgeLabelPosition, Background → White]}

(* Приклад 1. Ребра, що зустрічаються в графі автомата "AA". *)
Row[Graphics@DFARenderTransition[
     #, {{1, 0}, {2, 1}, {3, 0}, {4, 1}}, ArrowSize → .1] & /@
  DeleteDuplicates@DFACollapseEdges[DFAδEdges@DFA["AA"]⟦3⟧]⟦All, 2⟧]
```

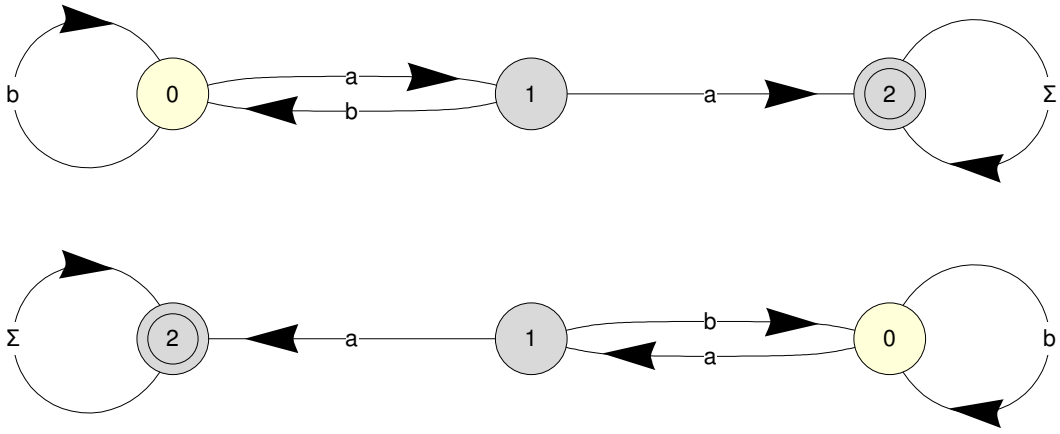


2.4 Зображення атоматів

Відмінність між функціями **GraphPlot** і **LayeredGraphPlot** полягає в тому, що перша має змогу промалювати графи багатьма різними алгоритмами, а друга розташовує вершини графа по шарах, розташовуючи більш домінантні вершини в позиції, що користувач може задати (**Left**, **Right**, **Top**, **Bottom**). Так, на основі цих функцій побудуємо відповідні їм функції промальовки ДСА. Основна частина коду функцій – це передача списку ребер, всіх опцій, та налаштування опцій промальовки вершин та ребер. Функції будують надписи за допомогою індексів станів та символів вибираючи їх відповідно із Q та Σ. По замовчуванню, у випадку об'єднання ребер, для простішого сприйняття надписи скорочуються до форми Σ\{…}, якщо список символів довший за половину алфавіту.

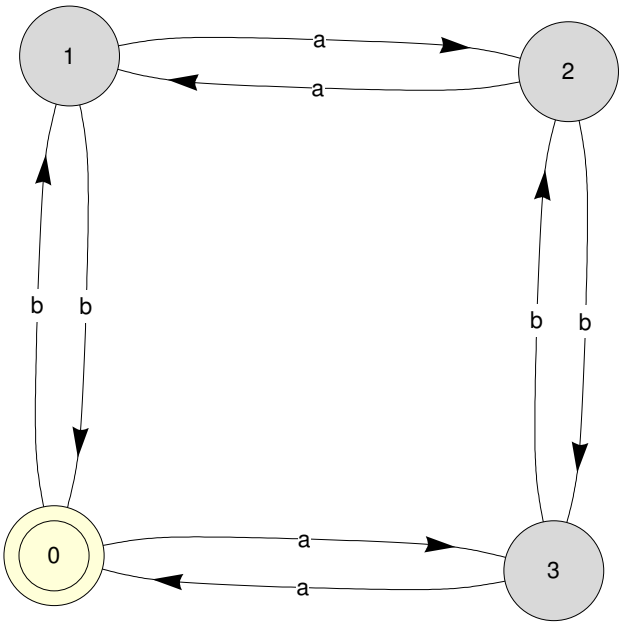
```
Options@DFACustomPlot =
  FilterRules[Options@GraphPlot, Except[#]] ~Join~ {EdgeCollapsing → True} ~
      Join~# &@@Options@DFARenderState ~Join~ Options@DFARenderTransition;
DFACustomPlot[{Q_, Σ_, δ_, s_, F_}, Edges_, PrePlot_,
  Plot_, opts : OptionsPattern[]] :=
 With[{Qf = If[Null =!= Q, Q⟦#⟧ &, Identity],
   Σf = If[Null =!= Σ, Σ⟦#⟧ &, Characters@FromCharacterCode[96 + #] &]},
  Plot @@ PrePlot@{If[OptionValue@EdgeCollapsing,
       DFACollapseEdges, Identity]@Edges@δ,
     Sequence @@ FilterRules[FilterRules[{opts}, Options@Plot],
       Except@{VertexRenderingFunction, EdgeRenderingFunction}],
     VertexRenderingFunction → Replace[OptionValue@
        VertexRenderingFunction, Automatic → (DFARenderState[
          #2, Replace[s, Null → 1], F, #1, VertexLabeling → Qf,
          Sequence @@ FilterRules[{opts}, Options@DFARenderState]] &)],
     EdgeRenderingFunction → Replace[OptionValue@
        EdgeRenderingFunction, Automatic →
        (DFARenderTransition[#3, #1, EdgeLabeling → If[Null =!= Σ, (l ↦ If[
                 Length@l == Length@Σ, "Σ", If[Length@l > Length@Σ / 2.,
                  Row@{"Σ", "\", Σf@Complement[Range@Length@Σ, l]},
                  Σf@l]]), FromCharacterCode[# + 96] &], Sequence @@
            FilterRules[{opts}, Options@DFARenderTransition]] &)]}]
DFAPlot[M_, opts : OptionsPattern[]] := DFACustomPlot[M,
  DFAδEdges, Identity, GraphPlot, opts]
DFALayeredPlot[M_, pos_: Left, opts : OptionsPattern[]] :=
 DFACustomPlot[M, DFAδEdges, Insert[pos, 2], LayeredGraphPlot, opts]
NFAPlot[M_, opts : OptionsPattern[]] :=
 DFACustomPlot[M, NFAδEdges, Identity, GraphPlot, opts]
NFALayeredPlot[M_, pos_: Left, opts : OptionsPattern[]] :=
 DFACustomPlot[M, NFAδEdges, Insert[pos, 2], LayeredGraphPlot, opts]
```

```
(* Приклад 1. Зобразимо автомат "AA". Функція
 DFAPlot промальовує його зліва направо,
а DFALayeredPlot може промалювати його справа наліво. *)
DFAPlot[DFA@"AA", ImageSize → 700]
DFALayeredPlot[DFA@"AA", Right, ImageSize → 700]
```

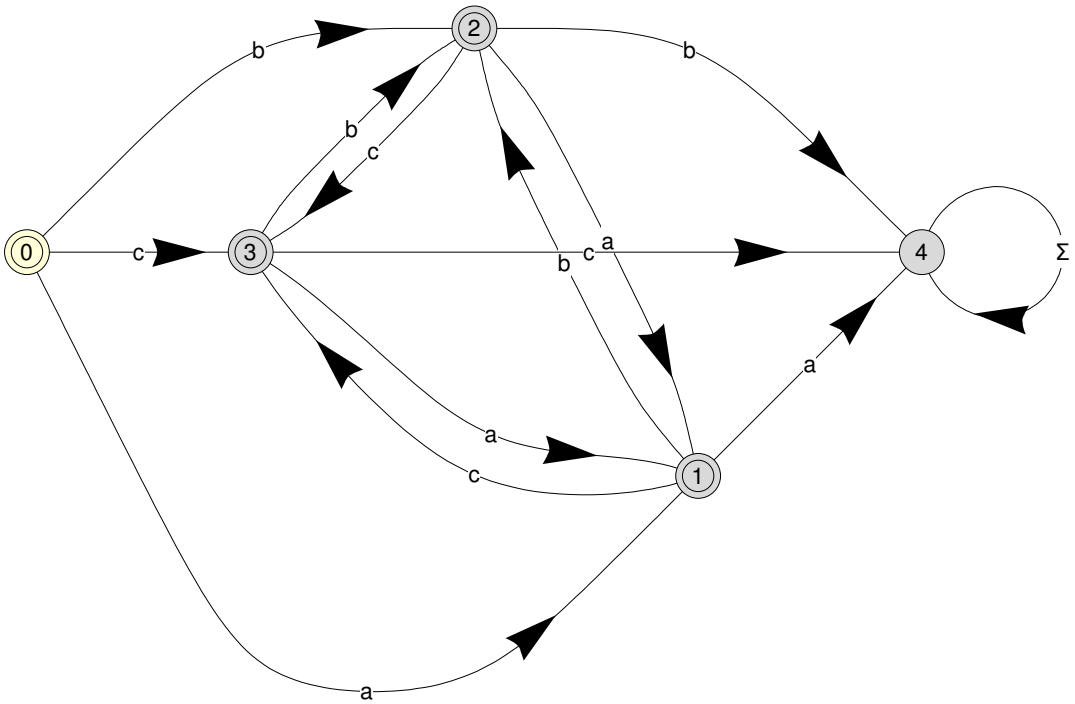



```
(* Приклад 2. Користувач може також задати заголовок зображення. *)
DFAPlot[DFA@#, ImageSize → 400, PlotLabel → #] &@"EvenEven"
DFALayeredPlot[DFA@#, Left, ImageSize → 700, PlotLabel → #] &@"DifABC"
```

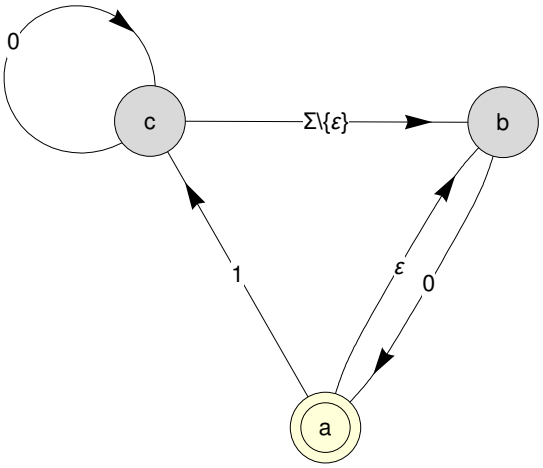
EvenEven



DifABC




```
(* Приклад 3. Зобразимо деякий εНСА. *)
NFAPlot[{{"a", "b", "c"}, {ε, 0, 1},
  {{{2}, {}, {3}}, {{}, {1}, {}}, {{}, {2, 3}, {2}}}, 1, {1}}]
```



3 Генерація ланцюжків мови ДСА

За допомогою заданого ДСА ми хочемо отримати список, що містить усі ланцюжки довжини n, що автомат допускає. Для цього побудуємо рекурсивну функцію з параметрами: M – автомат, q – стан, починаючи з якого M генеруватиме ланцюжки, n – довжина ланцюжків. Ця допоміжна функція реалізовуватиме наступний індуктивний алгоритм:

• **Базис.** Якщо $n == 0$, то починаючи із будь-якого заключного стану відомо, що M генерує ε (тобто пусту стрічку ""), інакше – нічого.

• **Індуктивний крок.** Якщо відомо, що M генерує ланцюжок ω довжиною n зі стану q і $(p, a\omega) \mapsto (q, \omega)$, тоді M генерує ланцюжок $a\omega$ довжиною $n + 1$ зі стану p.

Цією функцією буде функція **DFAAcceptedWordsIndex**, що повертатиме список індексів символів згенерованих ланцюжків. Приклад: нехай Σ = {"a", "b", "c"}, тоді список {1, 2, 1, 3} відповідатиме ланцюжку "abac".

Замість стану q використовуватимемо четвертий параметр автомата, тобто початковий стан s, що з кожним кроком рекурсії змінюватиметься.

```
(* Базис. Індекс ε - {}. *)
DFAAcceptedWordsIndex[{_, _, _, s_, F_}, 0] := If[MemberQ[F, s], {{}}, {}]
(* Індуктивний крок *)
DFAAcceptedWordsIndex[M : {_, _, δ_, s_, _}, n_] :=
 DFAAcceptedWordsIndex[M, n] = Module[{w, ans = {}},
   w = DFAAcceptedWordsIndex[ReplacePart[M, 4 → #], n - 1] & /@ δ[[s]];
   Do[If[Length@w[[i]] > 0,
     AppendTo[ans, Flatten@Prepend[#, i] & /@ w[[i]]]], {i, Length[w]}];
   Flatten[ans, 1]]

(* Приклад 1. Обчислимо всі індекси
 символів ланцюжків довжиною від 0 до 3 символів,
що генерує ДСА "EvenEven" і зобразимо їх в таблиці. *)
Grid[Table[Prepend[n]@DFAAcceptedWordsIndex[DFA@"EvenEvenAny", n],
  {n, 0, 3}], Dividers → {{False, True}}, Alignment → Left]
```

0	{}
1	{3}
2	{1, 1} {2, 2} {3, 3}
3	{1, 1, 3} {1, 3, 1} {2, 2, 3} {2, 3, 2} {3, 1, 1} {3, 2, 2} {3, 3, 3}

DFAAcceptedWords використовуватиме попередню функцію для формування стрічок.

```
DFAAcceptedWords[M_, n_] := Replace[StringJoin /@
    (Evaluate[ToString /@ M[[2]]][[#]] & /@ DFAAcceptedWordsIndex[M, n]), "" → {}]

(* Приклад 2. Обчислимо всі ланцюжки довжиною від 0 до 3 символів,
що генерують тестові ДСА і зобразимо їх в таблицях. *)
Row@Riffle[Grid[Prepend[{#[[1]], SpanFromLeft}]@
       Table[Prepend[n][DFAAcceptedWords[#[[2]], n] /. "" → "ε"], {n, 0, 3}],
      Frame → True, Dividers → {{True, True}, {True, True}},
      Alignment → Left] & /@ Normal@DFA, Spacer[5]]
```

OddA	
0	
1	a
2	ab ba
3	aaa abb bab bba

AA	
0	
1	
2	aa
3	aaa aab baa

SecondB	
0	
1	
2	ab bb
3	aba abb bba bbb

OneB	
0	
1	b
2	ab ba
3	aab aba baa

DifAB	
0	ε
1	a b
2	ab ba
3	aba bab

EvenEven	
0	ε
1	
2	aa bb
3	

Mod5	
0	
1	1
2	01
3	001 110

EvenEvenAny	
0	ε
1	c
2	aa bb cc
3	aac aca bbc bcb caa cbb ccc

DifABC	
0	ε
1	a b c
2	ab ac ba bc ca cb
3	aba abc aca acb bab bac bca bcb cab cac cba cbc

```
(* Приклад 3. Обчислимо числа у шістнадцятковій системі,
що діляться на 5 націло *)DFAAcceptedWords[ModMxN[16, 0, 5], 2]
{00, 05, 0a, 0f, 14, 19, 1e, 23, 28, 2d, 32, 37, 3c, 41, 46, 4b, 50, 55,
 5a, 5f, 64, 69, 6e, 73, 78, 7d, 82, 87, 8c, 91, 96, 9b, a0, a5, aa,
 af, b4, b9, be, c3, c8, cd, d2, d7, dc, e1, e6, eb, f0, f5, fa, ff}
```

4 Мінімізація ДСА

Кажуть, що два ДСА ***еквівалентні***, якщо вони задають одну й ту саму регулярну мову [1]. Для будь-якого ДСА існує еквівалентний ***мінімальний*** ДСА, такий, що серед всіх еквівалентних ДСА даний має мінімальну кількість станів (за теоремою Майхілла-Нероуда) [2, 4.4]. Насправді, такий мінімальний ДСА є унікальним з точністю до позначення станів (ми завжди можемо перейменувати назви станів, оскільки на мову автомата вони не впливають).

Мінімізація автоматів – важливий крок в будь-яких операціях над ними. Кількість надлишкових станів розмножується разом із застосуванням операцій, тож мінімізація необхідна.

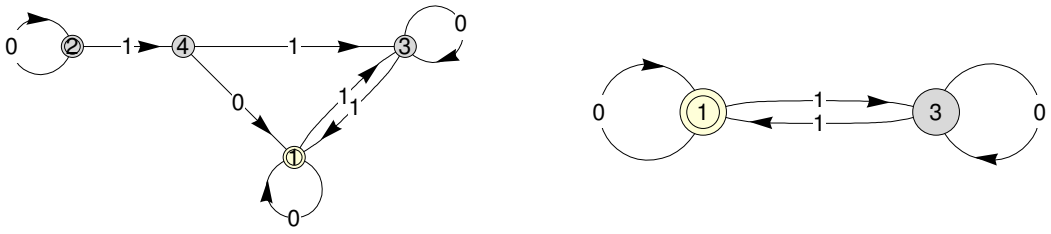
Процес мінімізації виконуватимемо в декілька кроків, описані в наступник пунктах.

4.1 Видалення недосяжних станів

Стан $q \in Q$ ***недосяжний***, якщо не існує конфігурації (s, ω), $\omega \in \Sigma^*$. Інакше кажучи, автомат ніколи не опиниться у недосяжному стані яким би вхідний ланцюжок не був.

Використаємо алгоритм визначення *зв'язної компоненти графа*, що реалізований вбудованою функцією **VertexOutComponent.** Після того, як отримаємо список індексів досяжних станів, у Q та δ залишимо лише ті елементи на які вказують індекси, а також виконаємо перетин із ними на F.

```
DFADeleteInaccesibleStates@{Q_, Σ_, δ_, s_, F_} :=
 {Q⟦#⟧, Σ}~Join~Insert[s, 2][{δ⟦#⟧, F⋂#} /. Thread[# → Range@Length@#]] &@
  VertexOutComponent[
   Graph@Flatten@Inner[DirectedEdge, Range@Length@δ, δ, List], s]

(* Приклад 1. Видалимо недосяжні стани невеликого ДСА. *)
Row[DFALayeredPlot[#, Left, ImageSize → Scaled@.49] & /@
    {#, DFADeleteInaccesibleStates@#} &@
  {Range@4, Range@2 - 1, {{1, 3}, {2, 4}, {3, 1}, {1, 3}}, 1, {1, 2}}]
```



4.2 Об'єднання еквівалентних станів

Стани $p \in Q$, $q \in Q$ ***еквівалентні***, якщо
$\nexists\, \omega \in \Sigma^* : \left(\exists\, (p_f, q_f) \in Q^2 : (q_f \in F \bigwedge q_f \notin F) \bigvee (q_f \notin F \bigwedge q_f \in F)\right)$.
Інакше кажучи, не існує ланцюжка, що розрізняє ці стани.

Щоб знайти еквівалентні класи застосуємо алгоритм "*табличного заповнення*", що рекурсивно визначає пари нееквівалентних класів, всі інші пари, що не знайдені цим алгоритмом є еквівалентними.

- **Базис.** $p \in F \bigvee q \in F \Rightarrow \{p, q\}$ – нееквівалентні.
- **Індуктивний крок.**

$\exists\, a \in \Sigma : \delta[\![\{p, q\}, a]\!]$ – нееквівалентні $\Rightarrow \{p, q\}$ – нееквівалентні.

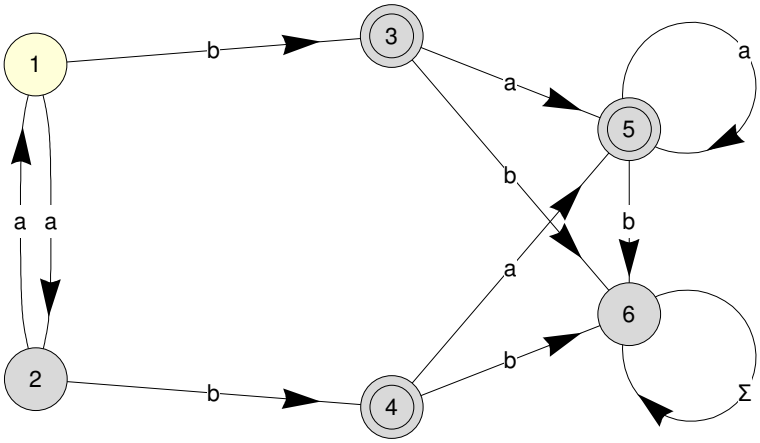
Опишемо функцію, що поверне список списків індексів станів, що є еквівалентними.

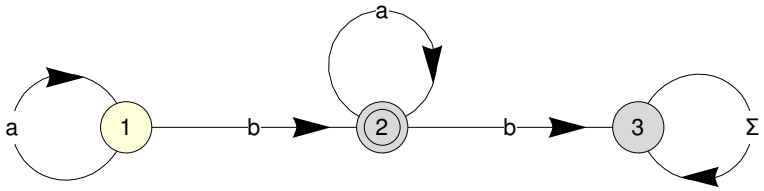
```
DFAEquivalenceClasses@{_, _, δ_, _, F_} := Module[{Q, Σ, P, C},
  {Q, Σ} = Range /@ Dimensions[δ, 2];
  P = (* побудувати усі пари *)Flatten[
    Subsets[Sort@#, {2}] & /@ (* забрати ті пари, що нееквівалентні
       за базисом *){Complement[Q, F], F} // Append[{Q, Q}^T], 1];
  P = (* вилучити нееквівалентні *)FixedPoint[
    DeleteCases[#, p_ /; MemberQ[Σ, a_ /; FreeQ[#, Sort@δ[[p, a]]]] &,
    P, SameTest → (Equal[Length /@ Unevaluated@##] &)];
  C = (* об'єднати *)P //. {h___, {a___, x_, b___}, u___,
      {p___, x_, q___}, t___} :↦ {h, {a, x, b, p, q}, u, t};
  C = (* додати відсутні стани *)SortBy[First]@Map[Union, #, {1}] &@
      Join[#, If[# ≠ {}, ArrayReshape[#, {Length@#, 1}], #] &@
        Complement[Q, Flatten@#]] &@C]

(* Приклад 1. Знайдемо еквівалентні класи ДСА "OneB". *)
DFAEquivalenceClasses@DFA@"OneB"

{{1, 2}, {3, 4, 5}, {6}}
```

4.3 Мінімізація

Останнім кроком є поєднання функцій видалення недосяжних станів і об'єднання еквівалентних станів в одну функцію. Зауважимо, що нам залишається сформувати нові Q, δ, s та F для мінімізованого автомата за допомогою списків еквівалентних класів.

```
DFAMinimize@{Q_, Σ_, δ_, s_, F_} := Module[{M, C},
  M = DFADeleteInaccesibleStates@{Q, Σ, δ, s, F};
  C = DFAEquivalenceClasses@M;
  {M[[1, #]] & /@ C, Σ} ~Join~ MapAt[Union, -1][{M[[3, C[[All, 1]]]], s, M[[5]]} /.
     Flatten@MapThread[Thread@*Rule, {C, Range@Length@C}]]]

(* Приклад 1. Мінімізуємо ДСА "OneB" і перейменуємо стани. *)
Column[DFAPlot[#, ImageSize → 500] & /@
    {#, MapAt[Range@*Length, 1]@DFAMinimize@#} &@DFA@"OneB"]
```





5 Операції над ДСА

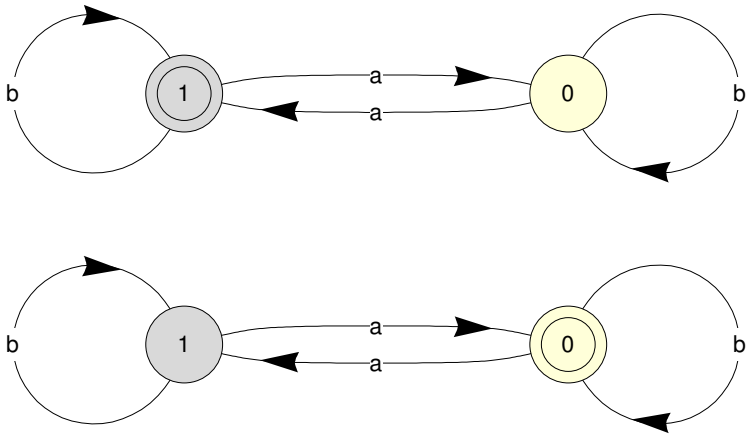
Згідно з теоремою Кліні регулярні мови замкнуті відносно булевих операцій, конкатенації та операції Кліні [2, 4.2]. В даному розділі опишемо ці операції над ДСА.

Зауваження. Для коректної роботи функцій, необхідно, щоб в ДСА $s == Q[\![1]\!]$.

5.1 Доповнення

Операція доповнення міняє місцями заключні й не заключні стани. Мова результуючого автомата – усі ланцюжки, що не допускав автомат-операнд.

```
DFAComplement@M_ := MapAt[Complement[Range@Length@M[[1]], #] &, -1]@M

(* Приклад 1. Мова ланцюжків з непарною кількістю символів
 "a" ⟶ мова ланцюжків з парною кількістю символів "a". *)
Column[DFAPlot[#, ImageSize → 500] & /@ {#, DFAComplement@#} &@DFA@"OddA"]
```

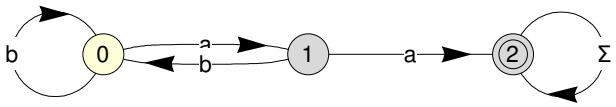


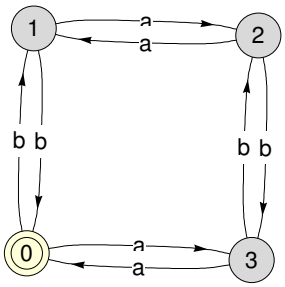
5.2 Об'єднання

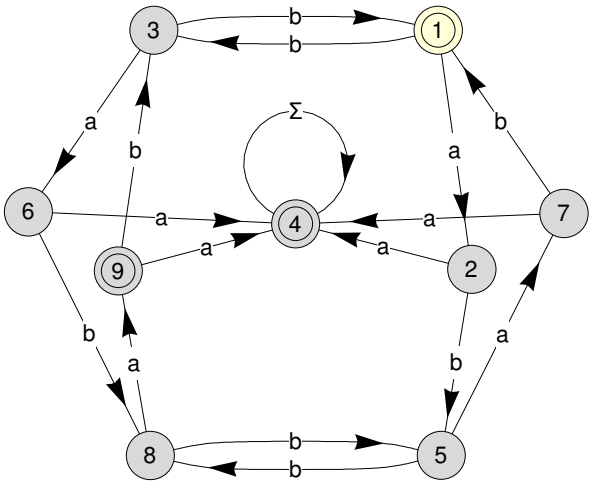
Операція об'єднання двох автоматів будує автомат, що допускає всі слова, що допускають автомати-операнди.

```
DFAUnion[A_, B_] := Module[{f, s, Q = {{A⟦4⟧, B⟦4⟧}}, δq, m = {}, rlQ},
  While[Q ≠ {}, {f, s} = Q⟦1⟧; δq = Thread@{A⟦3, f⟧, B⟦3, s⟧};
   AppendTo[m, {{f, s}, δq}]; Q = Complement[Join[Q, δq], m⟦All, 1⟧]];
  Q = m⟦All, 1⟧; rlQ = Range@Length@Q;
  MapAt[Range@*Length, 1]@
   DFAMinimize[{rlQ, DeleteDuplicates[A⟦2⟧ ~Join~ B⟦2⟧]} ~
      Join ~ {m⟦All, 2⟧, Q⟦1⟧, Select[Q,
        MemberQ[A⟦5⟧, #⟦1⟧] || MemberQ[B⟦5⟧, #⟦2⟧] &]} /. Thread[Q → rlQ]]]
```

```
(* Приклад 1. Автомат,
що сприймає ланцюжки з підстрічкою "aa" або ланцюжки із парною
 кількістю символів "a" та парною кількістю символів "b" *)Grid@
 {{DFAPlot[DFA@"AA", ImageSize → 400],
   DFAPlot[DFA@"EvenEven", ImageSize → 180]},
  {DFAPlot[DFAUnion[DFA@"AA", DFA@"EvenEven"], ImageSize → 380], …}}
```



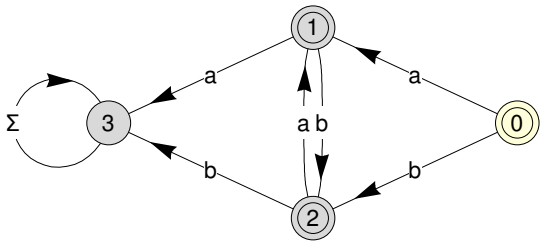


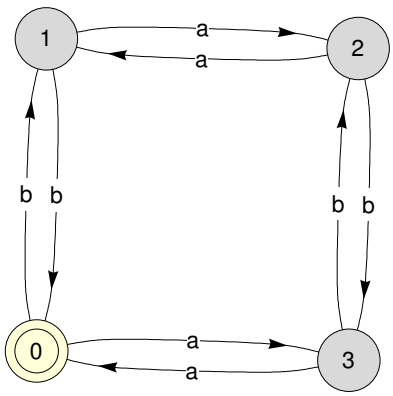


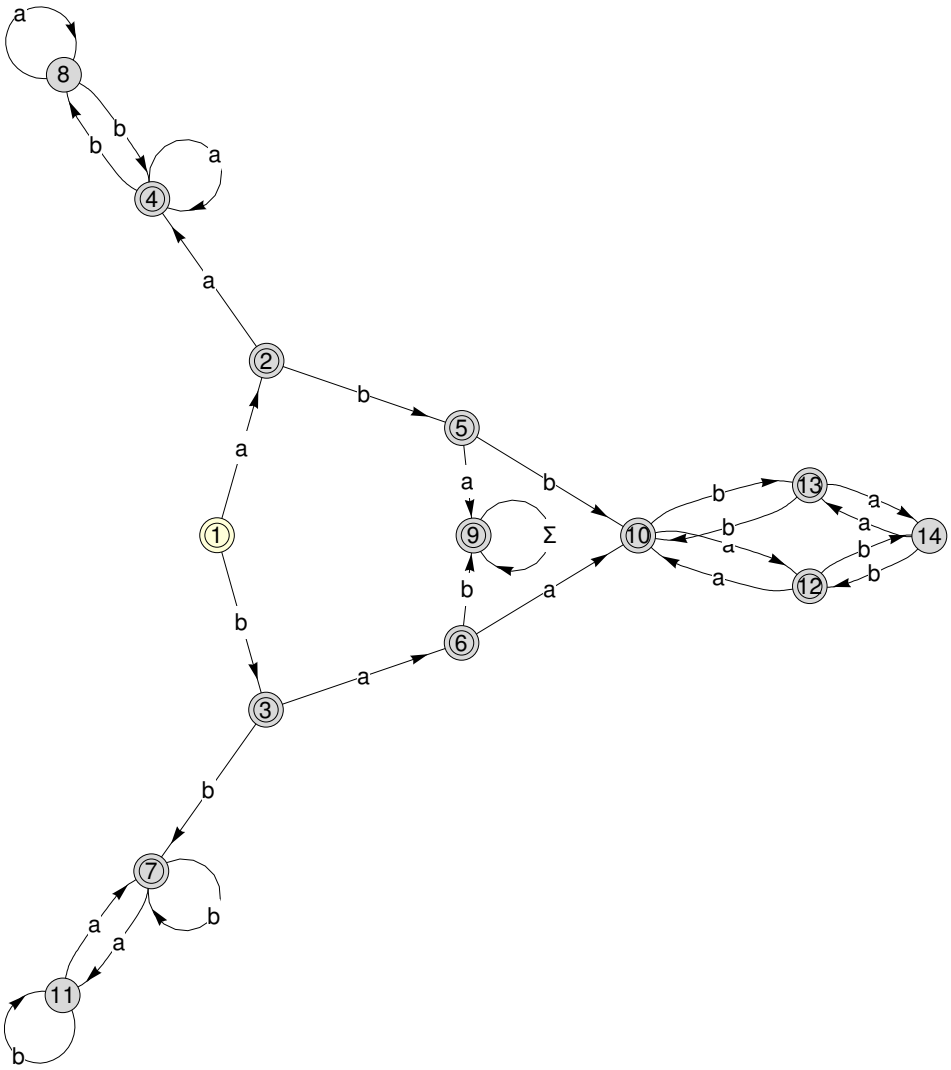
5.3 Конкатенація

Конкатенація двох автоматів будує автомат, що допускає лише конкатенацію ланцюжків автоматів-операндів.

```
DFAConcatenation[A_, B_] :=
 Module[{δ1, F1, δ2, F2, Qn, δ, M = {}, Q, x},
  {{δ1, F1}, {δ2, F2}} = {A, B}⟦All, {3, 5}⟧; Qn = Length@δ1;
  {δ2, F2} = {δ2, F2} + Qn; {δ1, δ2} = {δ1, δ2} /. x_Integer → {x};
  δ1 = δ1 /. (({#} → {#, Qn + 1}) & /@ F1);
  δ = Join[δ1, δ2] ; Q = {If[MemberQ[F1, 1], {1, Qn + 1}, {1}]};
  While[Q ≠ {}, Q = Q⟦1⟧;
   AppendTo[M, Join[{Q}, Sow[Union @@ # & /@ Thread@δ⟦Q⟧]]];
   Q = Select[Sow@Flatten[M, 1], FreeQ[M⟦All, 1⟧, #] &, 1]];
  Q = Select[M⟦All, 1⟧, (# ⋂ F2 ≠ {}) &];
  {δ1, Q} = {Rest /@ M, Q} /. Thread[M⟦All, 1⟧ → Range@Length@M];
  DFAMinimize@
   {Range@Length@δ1, DeleteDuplicates[A⟦2⟧ ~ Join ~ B⟦2⟧], δ1, 1, Q}]

(* Приклад 1. Автомат, що допускає конкатенацію двох ланцюжків: перший -
без двох однакових символів поруч, другий -
із парною кількістю символів і "a" і
 "b". Зауваження: один із ланцюжків або обидва можуть бути порожніми. *)
Row@{DFAPlot[DFA@"DifAB", ImageSize → 350],
  DFAPlot[DFA@"EvenEven", ImageSize → 250]}
DFAPlot[MapAt[Range@*Length, 1][
  DFAConcatenation[DFA@"DifAB", DFA@"EvenEven"]],
 ArrowSize → .015, MultiedgeStyle → .2, ArrowPosition → .83,
 Method → "SpringElectricalEmbedding", ImageSize → Full]
```



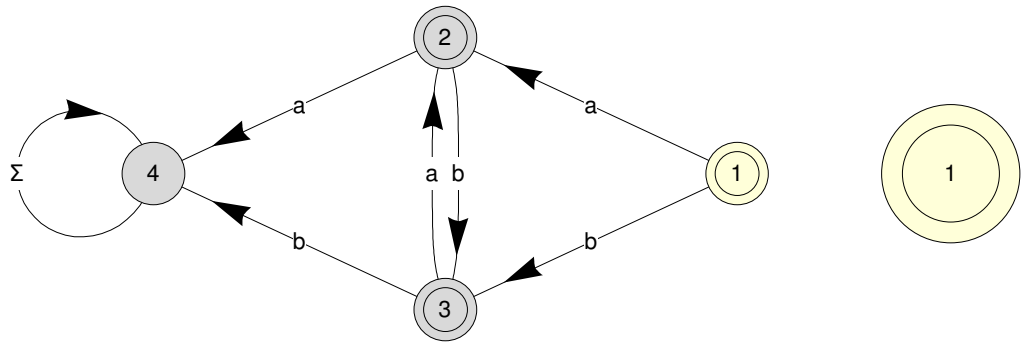



a
8
b
b
a
4
a
2
b
5
a
b
a
1
9
Σ
10
13
a
b
a
14
b
a
12
b
b
a
b
6
a
3
b
7
b
a
a
11
b

5.4 Замикання

Замикання автомата допускає замикання ланцюжків, що він допускав.

```
DFAClosure@m_ := Module[{δ, F, newM = {}, newRow, s = {{1}}, x},
  {δ, F} = m⟦{3, 5}⟧;
  F = Select[F, # > 1 &];
  δ = δ /. x_Integer → {x};
  δ = δ /. Map[({#} → {#, 1}) &, F] ;
  While[s ≠ {},
   s = First[s];
   AppendTo[newM, Join[{s}, Union@Flatten@# & /@ Thread@δ⟦s⟧]];
   s = Select[Flatten[newM, 1], FreeQ[newM⟦All, 1⟧, #] &, 1]];
  s = Select[newM⟦All, 1⟧, (# ⋂ F ≠ {}) &];
  {δ, s} = {Rest /@ newM, s} /.
    Thread[(#1 → #2) &[newM⟦All, 1⟧, Range@Length@newM]];
  If[FreeQ[m⟦5⟧, m⟦4⟧], {δ, s} = {δ, s} /. x_Integer → x + 1;
   PrependTo[δ, δ⟦1⟧]];
  DFAMinimize@{Range@Length@δ, m⟦2⟧, δ, m⟦4⟧, Join[{m⟦4⟧}, s]}]

(* Приклад 1. Очевидно, що замикання мови ланцюжків,
що не містять однакових символів поруч, є мова,
що містить будь-який ланцюжок. *)Row@MapAt[DFAPlot, 2]@
  MapAt[DFAPlot[#, ImageSize → 500] &, 1]@MapAt[Range@*Length, {All, 1}]@
    Through[{DFAMinimize, DFAClosure}@DFA@"DifAB"]
```



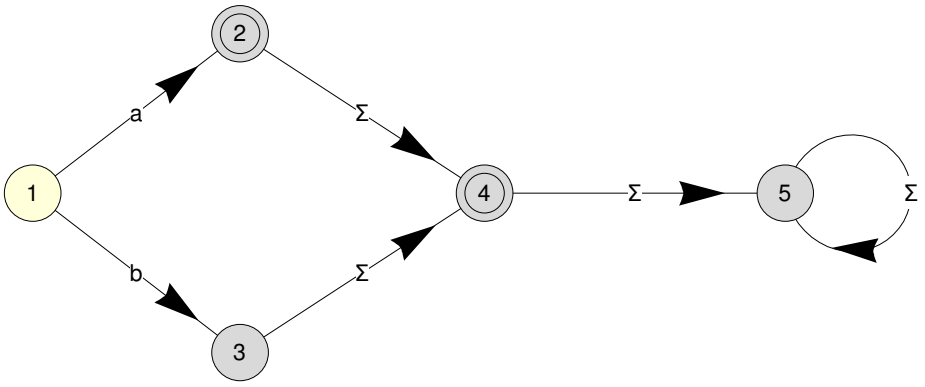
6 Деревоподібні ДСА

Очевидно, що діаграма переходів мінімального ДСА скінченної мови не містить циклів з яких досяжні заключні стани, оскільки як тільки вона містить хоча б один такий цикл, використовуючи його, ДСА може згенерувати ланцюжок як завгодно великої довжини. Такі ДСА скінченних мов називатимемо ***деревоподібними*** по аналогії із теорії графів, де *зв'язний ациклічний граф* називається *деревом*.

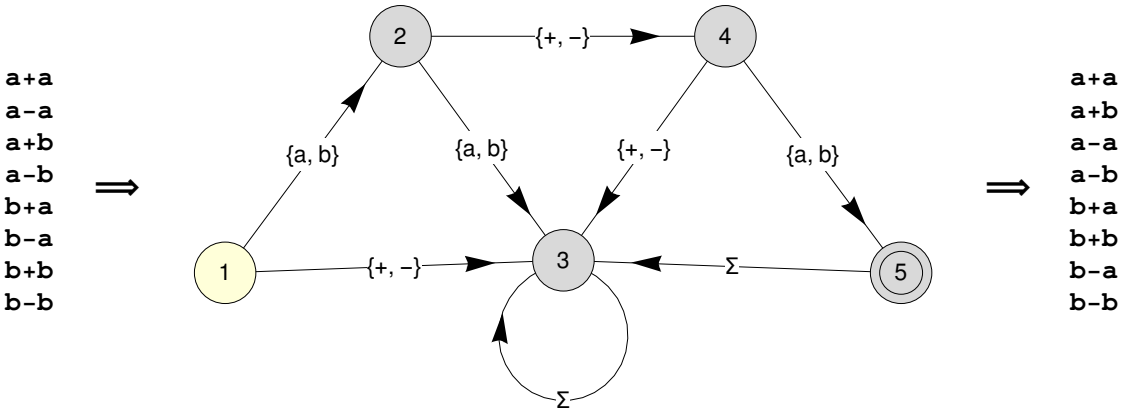
Опишемо функцію побудови ДСА, що буде поліморфною. Перший її аргумент – це список ланцюжків мови, а другий – або алфавіт, або лише кількість символів в ньому. В останньому випадку вважається, що алфавіт складається із символів "a", "b", "c", … [3, II, 9].

```
TreeDFA[{}, N_Integer] := {ConstantArray[1, N], {}}
TreeDFA[{"" | "ε"}, N_Integer] := {ConstantArray[2, {2, N}], {1}}
TreeDFA[{a_String, b__}, N_Integer] :=
 DFAUnion[TreeDFA[{a}, N], TreeDFA[{b}, N]]
TreeDFA[{a_String}, N_Integer] :=
 Module[{Qn = StringLength@a + 2, c = Characters@a, m},
  m = ConstantArray[Qn, {Qn, N}];
  Do[m⟦i, ToCharacterCode@c⟦i⟧ - 96⟧ = i + 1, {i, Length@c}];
  DFAMinimize@{Range@Length@m, CharacterRange[
      "a", FromCharacterCode[96 + Length@m⟦1⟧]], m, 1, {Qn - 1}}]
TreeDFA[{}, Σ_List] := {ConstantArray[1, Length@Σ], {}}
TreeDFA[{"" | "ε"}, Σ_List] := {ConstantArray[2, {2, Length@Σ}], {1}}
TreeDFA[{a_String, b__}, Σ_List] :=
 DFAUnion[TreeDFA[{a}, Σ], TreeDFA[{b}, Σ]]
TreeDFA[{a_String}, Σ_List] :=
 Module[{Qn = StringLength@a + 2, c = Characters@a, m},
  m = ConstantArray[Qn, {Qn, Length@Σ}];
  Do[m⟦i, First@FirstPosition[Σ, c⟦i⟧]⟧ = i + 1, {i, Length@c}];
  DFAMinimize@{Range@Length@m, Σ, m, 1, {Qn - 1}}]
```

```
(* Приклад 1. *)
DFAPlot[TreeDFA[{"a", "aa", "ab", "ba", "bb"}, 2], ImageSize → 600]
```




```
(* Приклад 2. Згенеруємо всі можливі двочлени над операціями
 додавання та віднімання використовуючи символи "a" та
 "b" як операнди. Побудуємо для них деревоподібний ДСА і
 перевіримо чи справді він генерує згенеровані двочлени. *)
Module[{𝔸 = Characters@"ab", 𝕆 = Characters@"+-", Σ, M}, Σ = 𝔸 ~Join~ 𝕆;
 Row@Flatten@Riffle[{
      Column[StringJoin @@@ Flatten[Through[((o ↦ Riffle[#, o] &) /@ 𝕆) @#] & /@
           Tuples[𝔸, {Length@𝕆}], 1]],
      DFAPlot[M = TreeDFA[StringJoin @@@ Flatten[Through[
                ((o ↦ Riffle[#, o] &) /@ 𝕆) @#] & /@ Tuples[𝔸, {Length@𝕆}], 1],
           Σ], ImageSize → 400, PlotRange → {{.1, 2.1}, {-.3, .8}}],
      Column@Flatten@Array[DFAAcceptedWords[M, #] &, 3, 2]},
     {{Spacer[20], Style["⟹", Bold, 20], Spacer[20]}}]]
```



7 Графічні інтерфейси побудови ДСА і деревоподібних ДСА

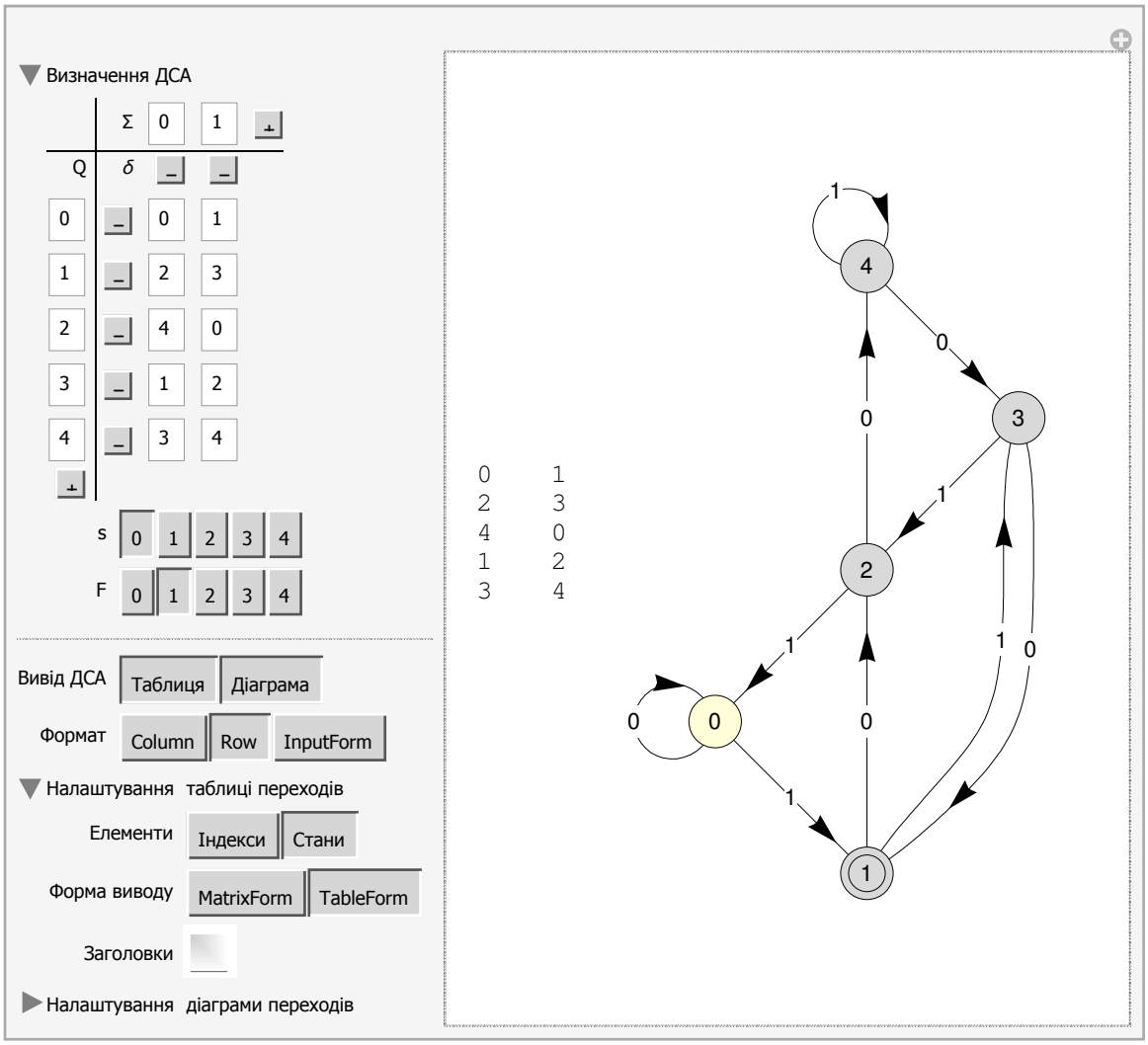
7.1 Зображення ДСА

На основі функції **Manipulate** і написаних функцій промальовування ДСА створимо візуальний інтерфейс, в якому користувач матиме змогу бачити зображення ДСА одразу після його опису. Налаштування міститимуть сам опис ДСА, вибір табличного чи графічного виводу та відповідні їм форми.

```
DFAManipulate@init_ := Quiet@
  With[{button = Function[, Button[##, ImageSize → {#, #} &@16,
       Appearance → "Palette", ContentPadding → False], {HoldAll}],
    inputField = Function[, InputField[##, FieldSize → {{1, 10}, {1, 10}}],
      {HoldAll}], outOptions = {"Таблиця", "Діаграма"}},
   Manipulate[
    (outForm /. {Column → (Column[#, Center] &)})[
     out /. {"Таблиця" :⇢ (Replace[δOutput,
           _ :> (δOutput[#, TableHeadings → {Q, Σ}] &) /; headingsQΣ])@
         (δElements /. {"Індекси" → Identity, "Стани" →
              ReplaceAll@(q_Integer :⇢ Q⟦q⟧)})@δ,
       "Діаграма" :⇢ plot[{Q, Σ, δ, s, F}, ImageSize → imageSize,
         PlotLabel → plotLabel, ArrowSize → arrowSize,
         ArrowPosition → arrowPosition, EdgeLabelPosition →
          edgeLabelPosition, VertexSize → vertexSize]}],
    {{Q, init⟦1⟧}, ControlType → None}, {{Σ, init⟦2⟧}, ControlType → None},
    {{δ, init⟦3⟧}, ControlType → None},
    OpenerView[{"Визначення ДСА", Dynamic@Grid[
       ArrayFlatten[{
         {{{}, "Σ"}, {"Q", "δ"}},
          Array[{inputField@Dynamic[Σ⟦#⟧], button["-",
                Catch@If[Length@Σ > 1, {Σ, δ} = {Drop[Σ, {#}],
                     Drop[δ, {}, {#}]}, Throw@Beep[]]]} &, Length@Σ]ᵀ,
          {{button["+", Catch[{Σ, δ} = {Append[Σ /. □ :⇢ Throw@Beep[], □],
                 ArrayFlatten@{{δ, {Range@Length@Q}ᵀ}}}]]}, {Null}}},
         {Array[{inputField@Dynamic[Q⟦#⟧, v ↦ Catch[{s, F} = {s, F} /.
                     Q⟦#⟧ → If[FreeQ[Q, v], Q⟦#⟧ = v, Throw@Beep[]]]],
              button["-", Catch@If[Length@Q > 1, {Q, δ} =
                   {Drop[Q, {#}], MapIndexed[{e, i} ↦ e /. Q⟦#⟧ → Q⟦i⟦1⟧⟧],
                     Drop[δ, {#}, {}] /. q_ :⇢ q - 1 /; q ≥ #]},
                 Throw@Beep[]]] &, Length@Q],
          Outer[inputField@Dynamic[Q⟦δ⟦##⟧⟧, v ↦ Catch[
```

```
                δ[[##]] = (FirstPosition[Q, v][[1]] /. "NotFound" :> Throw@
                      Beep[])]] &, Sequence @@ Range /@ Dimensions@δ],
          {ConstantArray[, Length@Q]}^T},
         {{{button["+",
             Catch[{Q, δ} = {Append[Q /. □ :> Throw@Beep[], □], Append[
                  δ, ConstantArray[Length@Q + 1, Length@Σ]]}]],}},
          {ConstantArray[, Length@Σ]},}}],
      Alignment → {, , {{1, 1} → {Center, Center}}},
      Dividers → {{False, True}, {False, True}}
     ]}, True],
{{s, init[[4]]}, Thread[Range@Length@Q → Q], ControlType → SetterBar},
{{F, init[[5]]}, Thread[Range@Length@Q → Q], ControlType → TogglerBar},
Delimiter,
{{out, outOptions[[{2}]], "Вивід ДСА"},
 outOptions, ControlType → TogglerBar},
{{outForm, Column, "Формат"}, {Column, Row, InputForm}},
Dynamic@Column[out /. {
     "Таблиця" → OpenerView[{"Налаштування  таблиці переходів", Column@{
          Control[{{δElements, "Індекси", "Елементи"},
            {"Індекси", "Стани"}}],
          Control[{{δOutput, MatrixForm, "Форма виводу"},
            {MatrixForm, TableForm}}],
          Control[{{headingsQΣ, False, "Заголовки"}, {True, False}}]
         }}],
     "Діаграма" →
      OpenerView[{"Налаштування  діаграми переходів", Column@{
          Control[{{plot, DFAPlot, "Функція"},
            {DFAPlot, DFALayeredPlot}}],
          Control[{{imageSize, Medium, "ImageSize"},
            {Tiny, Small, Medium, Large, Full, Automatic}}],
          Control[{{plotLabel, None, "PlotLabel"},
            "", FieldSize → {{6, 15}, {1, ∞}}}],
          Control[{{arrowSize, Automatic, "ArrowSize"}, 0., 1.}],
          Control[{{arrowPosition, .8, "ArrowPosition"}, 0., 1.}],
          Control[
           {{edgeLabelPosition, .5, "EdgeLabelPosition"}, 0., 1.}],
          Control[{{vertexSize, .1, "VertexSize"}, 0., 1.}]
         }}]}],
ControlPlacement → Left, SaveDefinitions → True]]
```

DFAManipulate@DFA@"Mod5"



DFAManipulate@ModMxN[4, 3, 6]

▼ Визначення ДСА

	Σ	0	1	2	3	⊥
Q	δ	_	_	_	_	
1	_	1	2	3	4	
2	_	5	6	1	2	
3	_	3	4	5	6	
4	_	1	2	3	4	
5	_	5	6	1	2	
6	_	3	4	5	6	
⊥						

s 1 2 3 4 5 6

F 1 2 3 4 5 6

Вивід ДСА Таблиця Діаграма

Формат Column Row InputForm

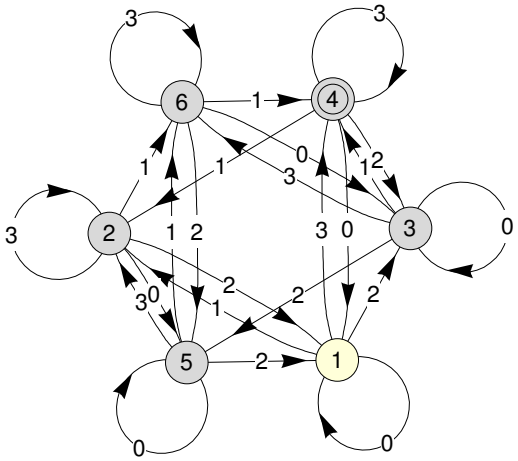
▶ Налаштування діаграми переходів

▼ Налаштування таблиці переходів

Елементи Індекси Стани

Форма виводу MatrixForm TableForm

Заголовки ☑



	0	1	2	3
1	1	2	3	4
2	5	6	1	2
3	3	4	5	6
4	1	2	3	4
5	5	6	1	2
6	3	4	5	6

7.2 Зображення деревоподібних ДСА

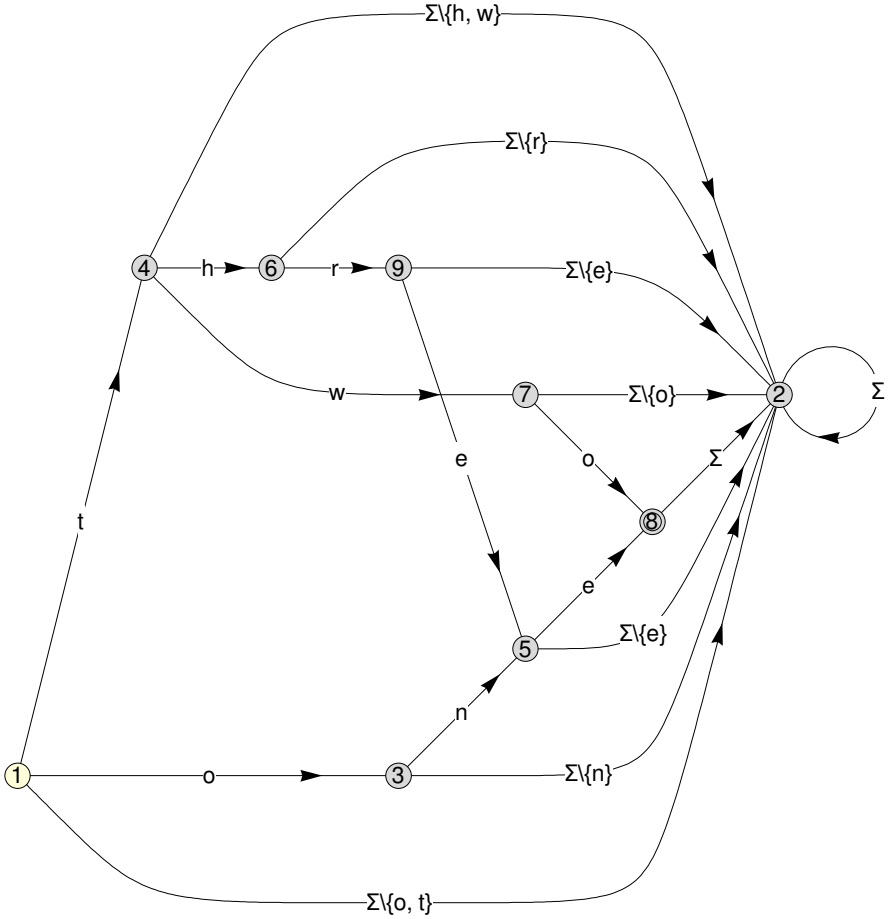
Створимо графічний інтерфейс побудови деревоподібних ДСА використавши інший підхід, не використовуючи **Manipulate**. Для цього створимо динамічний модуль з полями для вводу слів, виводом ДСА у табличній формі та опцією виводу діаграми.

```
With[{input =
    Function[, InputField[#, String, FieldSize → {{4, ∞}, 1}], {HoldAll}],
  button = Function[, Button[Style[#1, Gray, 16],
      #2, Appearance → "Frameless"], {HoldAll}]},
 DynamicModule[{W = StringSplit@"one two three", A,
   w = "", showPlot = False}, Column[
   {Dynamic@Row@Join[{"Слова:", Spacer[10]}, Array[Superscript[input[
            Dynamic[W[[#]], v ↦ If[StringLength@v > 0, W[[#]] = v, Beep[]]]],
          button["×", If[Length@W > 1, W = Drop[W, {#}], Beep[]]] &,
        Length@W], {Spacer[10], Superscript[input@Dynamic@w, button["+",
          If[w =!= "" && FreeQ[W, w], AppendTo[W, w] (w = ""), Beep[]]]]}],
    Dynamic[TableForm[#[[3]], TableHeadings → {MapAt[
             Row@{"*", Spacer@5, #} &, ArrayReshape[#, {Length@#, 1}] &@
              #[[5]]]@MapAt[Row@{"→", Spacer@5, #} &, #[[4]]@#[[1]], #[[2]]},
        TableAlignments → {Right}] &[A = TreeDFA[W,
        Union @@ Characters@W]]],
    Row@{Checkbox[Dynamic[showPlot]], Spacer@5, "Малювати діаграму"},
    Dynamic@
     If[showPlot, DFALayeredPlot[TreeDFA[W, Union @@ Characters@W], Left,
       ImageSize → Large, ArrowSize → .02], ""]}], SaveDefinitions → True
]]
```

Слова: one × two × three × +

	e	h	n	o	r	t	w
→ 1	2	2	2	3	2	4	2
2	2	2	2	2	2	2	2
3	2	2	5	2	2	2	2
4	2	6	2	2	2	2	7
5	8	2	2	2	2	2	2
6	2	2	2	2	9	2	2
7	2	2	2	8	2	2	2
* 8	2	2	2	2	2	2	2
9	5	2	2	2	2	2	2

☑ Малювати діаграму



Висновки

У роботі розроблений програмний функціонал операцій над скінченними автоматами, що дозволяє на його основі створювати різноманітні програми, що полегшують вивчення теорії автоматів.

Створені графічні інтерфейси для зображення ДСА, побудови деревоподібних ДСА, які використовуються в курсах “Системне програмування” та “Теорія компіляції” на третіх та четвертих курсах, для наглядної ілюстрації програмного матеріалу.

Робота заохочує читача покращувати, доповнювати та використовувати написані функції у дослідженні скінченних автоматів.

Перелік посилань

1. Сопронюк Т.М. Системне програмування. Частина І. Елементи теорії формальних мов: Навчальний посібник у двох частинах. - Чернівці: “Рута”, 2008. - 84 с.
2. Hopcroft J.E., 1939- Introduction to automata theory, languages and computation. 2nd ed. / John E. Hopcroft, Rajeev Motwani, Jaffrey D. Ullman. – Addison-Wesley, 2001. - 521 p.
3. http://library.wolfram.com/infocenter/*MathSource*/892/ – Jaime Rangel-Mondragón. Algorithms on Finite Automata, parts I - IV