

Tworzenie automatów z wyrażeń regularnych metodą Thompsona

1 Cel ćwiczenia

Celem ćwiczenia jest ugruntowanie znajomości programów `flex` i `bison`, utrwalenie umiejętności dokonywania analizy składniowej oraz rozszerzenie wiadomości na temat automatów skończonych.

2 Środowisko

Składnia polecenia `dot`:

```
dot -Tps < źródło > wynik.ps
```

Najszybciej można uzyskać wynik pisząc np.:

```
echo '0(0|1)*0' | ./z7a | dot -Tps > wynik.ps; gv wynik.ps &
```

3 Metoda Thompsona

Automat tworzony jest z części składowych w podobny sposób, w jaki oblicza się wartość wyrażenia arytmetycznego. Cechą charakterystyczną metody Thompsona jest to, że powstający automat ma zawsze nie tylko jeden stan początkowy, ale i **jeden stan końcowy**. W kalkulatorze podstawowymi elementami wyrażenia były liczby całkowite i rzeczywiste oraz komórki pamięci (M_1, M_2, \dots). W wyrażeniu regularnym podstawowymi elementami są: zbiór pusty \emptyset , pusty ciąg symboli ε , oraz symbol σ z alfabetu Σ .

Pustemu zbiorowi \emptyset odpowiada automat ze stanem początkowym i stanem końcowym, ale bez przejść. **Pustemu ciągowi symboli** ε odpowiada automat ze stanem początkowym i stanem końcowym oraz z przejściem między tymi stanami etykietowanym ε . **Symbolowi z alfabetu** $\sigma \in \Sigma$ odpowiada automat ze stanem początkowym i stanem końcowym oraz z przejściem między tymi stanami etykietowanym tym symbolem.

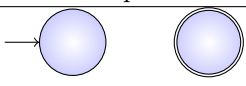
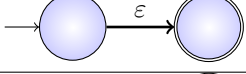
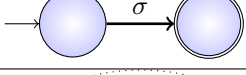
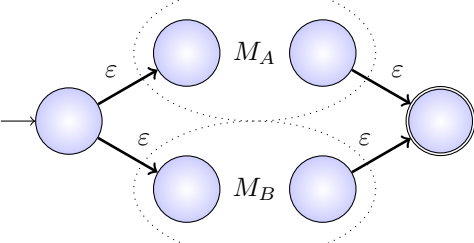
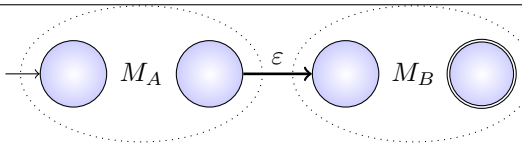
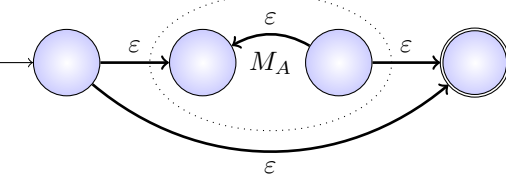
Mając podstawowe cegiełki, można tworzyć większe automaty korzystając ze sklejania wyrażeń, alternatywy i domknięcia przechodniego. Niech M_A i M_B będą automatami rozpoznającymi odpowiednio wyrażenia R_A i R_B . Automat dla **sklejania** $R_A R_B$ wyrażeń R_A i R_B tworzymy czyniąc stan końcowy automatu M_A niekończącym i łącząc go przejściem etykietowanym ε ze stanem początkowym automatu M_B . Stanem początkowym automatu wynikowego będzie stan początkowy automatu M_A , a stanem końcowym – stan końcowy automatu M_B .

Automat dla **alternatywy** $R_A | R_B$ wyrażeń R_A i R_B tworzymy dodając nowy stan początkowy i nowy stan końcowy. Dodajemy też cztery przejścia etykietowane ε : dwa od nowego stanu początkowego do stanów początkowych automatów M_A i M_B , dwa od stanów końcowych automatów M_A i M_B do nowego stanu końcowego. Stany końcowe automatów M_A i M_B przestają być końcowe.

Automat dla **domknięcia przechodniego** R_A^* wyrażenia R_A tworzymy dodając nowy stan początkowy i nowy stan końcowy. Dodajemy też cztery nowe przejścia etykietowane ε : z nowego stanu początkowego do stanu początkowego automatu M_A , z nowego stanu początkowego do nowego stanu końcowego, ze stanu końcowego automatu M_A do stanu początkowego automatu M_A i ze stanu końcowego automatu M_A do nowego stanu końcowego. Stan końcowy automatu M_A przestaje być końcowym.

W wyrażeniu regularnym można też stosować **nawiasy** do grupowania na podobnym zasadach, jak w wyrażeniach arytmetycznych. Konstrukcję Thompsona podsumowuje tabela na następnej stronie, zaś dodatkowe informacje na temat przedstawionych zagadnień można znaleźć w książkach:

- John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, *Wprowadzenie do teorii automatów, języków i obliczeń*, Wydawnictwo Naukowe PWN, Warszawa, 2005;
- Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, *Kompilatory. Reguły, metody i narzędzia*, Wydawnictwo Naukowo-Techniczne, seria *Klasyka informatyki*, Warszawa, 2002;

	Thompson
$\emptyset \in RE$	
$\varepsilon \in RE$	
$\sigma \in \Sigma \Rightarrow \sigma \in RE$	
$R_A, R_B \in RE \Rightarrow R_A R_B \in RE$	
$R_A, R_B \in RE \Rightarrow R_A R_B \in RE$	
$R_A \in RE \Rightarrow R_A^* \in RE$	

4 Dostarczony program

W pliku `z7a.tgz` znajduje się szkielet programu do tworzenia automatów na podstawie wyrażeń regularnych. Zawiera pełny analizator leksykalny i fragment analizatora składniowego. Należy w nim uzupełnić analizator składniowy. Wejściem programu jest tekst wyrażenia regularnego, gdzie $\Sigma = \{0, \dots, 9, a, \dots, z\}$. Wyjściem jest plikiem w formacie programu dot z pakietu graphviz dostępnego pod adresem <http://www.graphviz.org/>. Na wyjściu są diagramy przejść stanów dla trzech automatów: automatu niedeterministycznego (wynik konstrukcji Thompsona), deterministycznego (automat Thompsona po determinizacji) i minimalnego automatu deterministycznego.

Uzupełnienia wymagają jedynie reguły składniowe umieszczone pomiędzy parami znaków `%%` w programie analizatora składniowego. Dostarczono jedynie regułę rozpoznającą symbol alfabetu lub pusty ciąg symboli ε . Należy dodać reguły dla pozostałych konstrukcji. Do tworzenia stanu służy funkcja `create_state`. Nie ma parametrów. Wynikiem jest numer utworzonego stanu. Do tworzenia przejścia służy funkcja `create_transition`. Jej pierwszym parametrem jest numer stanu źródłowego, drugim – numer stanu docelowego, trzecim – etykieta przejścia. Aby przejście było etykietowane ε , należy jako etykiety użyć stałej `EPSILON`.

W celu składania większych wyrażeń z mniejszych musimy znać numery stanów początkowych i końcowych podwyrażeń. Aby uniknąć kłopotów z przekazywaniem struktur w czystym języku C, stan początkowy i stan końcowy zapisywane są w tablicy `subRes`. Dla bieżącego (właśnie tworzonego) wyrażenia należy do tego użyć funkcji `createRE`. Wynikiem tej funkcji jest indeks elementu tablicy `subRes`. Należy go przekazać jako wynik konstrukcji składniowej w zmiennej `$$` na końcu obsługi wyrażenia:

```
$$ = createRE(początkowy,końcowy); /* początkowy i końcowy stan bieżącego RE */
```

Do wyłuskania stanu początkowego i stanu końcowego automatu odpowiadającego wyrażeniu służą makra `FIRST()` i `LAST()`, np. jeśli chcemy określić numer stanu początkowego automatu dla trzeciego wyrażenia w regule produkcji konstrukcji składniowej, piszemy:

FIRST(\$3) /* dla stanu końcowego wymieniamy „FIRST” na „LAST” */

5 Zadanie do wykonania

1. Dopisać obsługę wyrażenia pustego \emptyset i wyrażenia w nawiasach.
2. Dopisać obsługę alternatywy.
3. Dopisać obsługę sklejania. Wykorzystać priorytet operatora CONCAT (sklejenie to dwa wyrażenia po sobie; nie jest potrzebny dodatkowy operator).
4. Dopisać obsługę domknięcia przechodniego.
5. Dopisać rozszerzenie: operator domknięcia „+”. Tu należy także uzupełnić analizator leksykalny i ustalić priorytet.

Jeśli dla wyrażenia regularnego z przykładu powstają inne automaty niż podane na rysunkach, należy wypróbować działanie programu na najprostszych wyrażeniach typu „01” „0|1” „0*” i porównać wynik z rysunkami w tabelce. Jeśli jest OK, błędy mogą być w parametrach funkcji `createRE` – określeniu stanu początkowego i końcowego. Automaty mogą mieć inną numerację stanów – wtedy wynik jest poprawny.

6 Przykład

Wyrażenie: `0(0|1)*0` (ciąg zer i jedynek zaczynający się i kończący zerem).

Wynik w postaci tekstowej:

```
digraph "0(0|1)*0" {
    rankdir=LR;
    node[shape=circle];

    subgraph "clustern" {
        color=blue;
        n11 [shape=doublecircle];
        n [shape=plaintext, label=""]; // dummy state
        n -> n0; // arc to the start state from nowhere
        n0 -> n1 [label="0"];
        n2 -> n3 [label="0"];
        n4 -> n5 [label="1"];
        n6 -> n2 [fontname="Symbol", label="e"];
        n6 -> n4 [fontname="Symbol", label="e"];
        n3 -> n7 [fontname="Symbol", label="e"];
        n5 -> n7 [fontname="Symbol", label="e"];
        n7 -> n6 [fontname="Symbol", label="e"];
        n8 -> n6 [fontname="Symbol", label="e"];
        n7 -> n9 [fontname="Symbol", label="e"];
        n8 -> n9 [fontname="Symbol", label="e"];
        n10 -> n11 [label="0"];
        n9 -> n10 [fontname="Symbol", label="e"];
        n1 -> n8 [fontname="Symbol", label="e"];
        label="NFA"
    }

    subgraph "clusterd" {
        color=blue;
        d2 [shape=doublecircle];
        d [shape=plaintext, label=""]; // dummy state
```

