Tworzenie automatów z wyrażeń regularnych

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
```

Najlepiej testować tworzony program pisząc poniższy potok z własnym wyrażeniem jako argumentem polecenia `echo`:

```
echo '0(0|1)*0' | ./z7c | dot -Tps > wynik.ps; gv wynik.ps &
```

3 Metoda Yamada-McNaughton-Głuszkow

Automat niedeterministyczny $M = (Q, \Sigma, \delta, q_0, F)$, gdzie Q to zbiór *stanów*, Σ to skończony zbiór symboli zwany *alfabetem*, $\delta : Q \times \Sigma \to 2^Q$ to *funkcja przejścia*, $q_0 \in Q$ to stan początkowy, zaś $F \subseteq Q$ to zbiór stanów końcowych, tworzony jest z części składowych w podobny sposób, w jaki oblicza się wartość wyrażenia arytmetycznego. Cechą charakterystyczną automatów Głuszkowa powstających w tej metodzie jest to, że wszystkie przejścia wchodzące do danego stanu mają tę samą etykietę. Przy tworzeniu automatu numeruje się wszystkie symbole występujące w wyrażeniu regularnym, np. w wyrażeniu „010" pierwsze zero jest różne od drugiego; całe wyrażenie jest traktowane jako $0_1 1_2 0_3$. Wykorzystuje się cechę *Null* oraz zbiory *First*, *Last* i *Follow*. Cecha *Null* jest równa prawdzie, jeśli pusty ciąg symboli jest rozpoznawany przez to wyrażenie, tzn. dla wyrażenia regularnego $r \in RE$:

$$Null(r) \Leftrightarrow (\varepsilon \in \mathcal{L}(r)) \tag{1}$$

Zbiór *First* to zbiór numerowanych symboli, które mogą się pojawiać na pierwszym miejscu słów należących do języka danego wyrażenia, tzn. dla $r \in RE$:

$$First(r) = \{\sigma \in \Sigma : (\exists w \in \Sigma^*)\sigma w \in \mathcal{L}(r)\} \tag{2}$$

Zbiór *Last* to zbiór numerowanych symboli, które mogą się pojawiać na ostatnim miejscu słów należących do języka danego wyrażenia, tzn. dla $r \in RE$:

$$Last(r) = \{\sigma \in \Sigma : (\exists w \in \Sigma^*)w\sigma \in \mathcal{L}(r)\} \tag{3}$$

Zbiór *Follow* to zbiór par numerowanych symboli, które mogą pojawiać się w takiej kolejności jeden bezpośrednio po drugim w słowach należących do języka danego wyrażenia, tzn. dla $r \in RE$:

$$Follow(r) = \{(\sigma_1, \sigma_2) \in \Sigma^2 : (\exists u, w \in \Sigma^*)u\sigma_1\sigma_2 w \in \mathcal{L}(r)\} \tag{4}$$

Wartości *Null*, *First*, *Last* i *Follow* dla podstawowych wyrażeń $\emptyset$, ε i $\sigma \in \Sigma$ określa tabela 1. Mając dane dwa wyrażenia $r_1 \in RE$ i $r_2 \in RE$ oraz obliczone dla nich wartości *Null*, *First*, *Last* i *Follow* możemy policzyć te wartości dla wyrażeń $r_1|r_2$ (alternatywa), $r_1 r_2$ (sklejenie) i r_1^* (domknięcie przechodnie) zgodnie z tabelą 1.

Mając wartości *Null*, *First*, *Last* i *Follow* dla całego wyrażenia można zbudować automat. Każdy numerowany symbol związany jest z osobnym stanem, dlatego **dla każdego symbolu** w wyrażeniu tworzony jest **stan automatu** i w stanie zapamiętywany jest symbol. Numer stanu służy potem jako numerowany symbol.

Ponieważ symbole związane są ze stanami docelowymi, należy utworzyć jeden dodatkowy stan, który będzie służył jako **stan początkowy**. Stan ten będzie stanem końcowym, jeśli wartość *Null* dla całego wyrażenia będzie wynosić *true*.

W zbiorze *First* zapamiętane są numery stanów, do których biegną **przejścia ze stanu początkowego**. Przejścia etykietowane są symbolami związanymi ze stanami docelowymi.

RE	*Null*	*First*	*Last*	*Follow*
$\emptyset$	*false*	$\emptyset$	$\emptyset$	$\emptyset$
ε	*true*	$\emptyset$	$\emptyset$	$\emptyset$
$\sigma \in \Sigma$	*false*	$\{\sigma\}$	$\{\sigma\}$	$\emptyset$
$r_1 \vert r_2$	$Null(r_1) \vee Null(r_2)$	$First(r_1) \cup First(r_2)$	$Last(r_1) \cup Last(r_2)$	$Follow(r_1) \cup Follow(r_2)$
$r_1 r_2$	$Null(r_1) \wedge Null(r_2)$	$First(r_1)$ *if* $\neg Null(r_1)$ *else* $First(r_1) \cup First(r_2)$	$Last(r_2)$ *if* $\neg Null(r_2)$ *else* $Last(r_1) \cup Last(r_2)$	$Follow(r_1) \cup Follow(r_2) \cup (Last(r_1) \times First(r_2))$
r_1^*	*true*	$First(r_1)$	$Last(r_1)$	$Follow(r_1) \cup (Last(r_1) \times First(r_1))$

Tablica 1: Cecha Null oraz zbiory First, Last i Follow dla poszczególnych typów wyrażeń regularnych

Numery **stanów końcowych** (poza stanem początkowym, którego końcowość zależy od cechy *Null*), zapamiętane są w zbiorze *Last*.

Przejścia prowadzące ze stanów innych niż początkowy pamiętane są w zbiorze *Follow*. Pierwszy element przechowywanych tam par określa stan źródłowy przejścia, drugi – stan docelowy. Etykieta określona jest poprzez stan docelowy.

Więcej informacji na temat tworzenia automatów z wyrażeń regularnych można znaleźć w następujących pozycjach:

- 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;
- H.M.M. ten Eikelder, H.P.J. van Geldrop, *On the correctness of some algorithms to generate finite automata for regular expressions*, Department of Mathematics and Computing Science, Eindhoven University of Technology, Computing Science Note 93/32, Eindhoven, September 1993.

4 Dostarczony program

W plikach `z7c.y` i `z7c.l` 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 $\Sigma = \{0, \ldots, 9, a, \ldots, z\}$. Wyjście 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 Yamada-McNaughton-Głuszkow), deterministycznego (automat Głuszkowa 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. Reguła obsługująca symbol należący do alfabetu tworzy nowy stan wywołując funkcję `create_state()`. Ta funkcja nie jest wykorzystywana w żadnych innych regułach – używa się jej jeszcze tylko w funkcji `create_NFA()` do utworzenia stanu początkowego automatu.

Każda reguła tworzącą nowy automat dla nowego podwyrażenia (nie wszystkie podwyrażenia wymagają utworzenia nowego automatu) musi wywołać w tym celu funkcję `createRE()`. Parametrami funkcji są: cecha *Null*, zbiór stanów (numerowanych symboli) *First*, zbiór stanów *Last*, zbiór par stanów *Follow*. Wynikiem funkcji jest krotka zawierająca cechę *Null* oraz zbiory *First*, *Last* i *Follow*. Użytkownik nie ma do nich bezpośredniego dostępu. Jeżeli `x` jest wynikiem funkcji `createRE()`, to cechę *Null* wyłuskujemy wywołując funkcję `RE_NULL(x)`, zbiór *First* wywołując funkcję `RE_FIRST(x)`, zbiór *Last* — funkcję `RE_LAST(x)` i zbiór *Follow* — funkcję `RE_FOLLOW(x)`.

Cecha *Null* może przyjmować dwie wartości: `FALSE` i `TRUE`. Taki jest też wynik funkcji `RE_NULL()`. Na wartościach *Null* można stosować operatory języka C `&&` i `||`.

Zbiory *First* i *Last* są początkowo albo puste oznaczone za pomocą stałej `EMPTY_FIRST_OR_LAST_SET`, albo tworzone są w regule dla pojedynczego symbolu (i tylko tam) za pomocą funkcji `create_set()`, której parametrem jest stan (numerowany symbol) uzyskany z funkcji `create_state()`. Zbiory z większą liczbą elementów tworzone są za pomocą funkcji sumy zbiorów `merge_sets()`, której oba parametry są zbiorami *First* lub *Last*.

Zbiory par stanów *Follow* są początkowo albo puste oznaczone za pomocą stałej `EMPTY_FOLLOW_SET`, albo tworzone są za pomocą funkcji iloczynu kartezjańskiego dwóch zbiorów stanów `set_product()`. Zbiory par stanów można dodawać za pomocą funkcji `merge_follow_sets()`.

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). Przykład ustalania priorytetu operacji z wykorzystaniem wirtualnego operatora (analizator leksykalny nigdy nie zwraca wartości CONCAT) można znaleźć wywołując polecenie `info bison` (lub `Ctrl-H i` w emacsie), przechodząc do `Examples` i następnie do `Infix Calc`.
4. Dopisać obsługę domknięcia przechodniego.
5. Dopisać rozszerzenie: operator domknięcia „+”. Należy uzupełnić analizator leksykalny i ustalić priorytet. Uwaga: ε^+ czy $((abc)^*(\varepsilon|def))^+$ są poprawne.

Jeśli dla wyrażenia z przykładu powstają inne automaty niż podane na rysunkach, należy wypróbować działanie programu na najprostszych wyrażeniach typu „01” „0|1” „0*”. Jeśli jest OK, błędy mogą być w parametrach funkcji `createRE`. Częsty błąd polega na użyciu zmiennej $2 zamiast $3, gdy $2 oznacza „wartość” operatora.

6 Przykład

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

Wynik w postaci tekstowej:

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

  subgraph "clustern" {
    color=blue;
    n3 [shape=doublecircle];
    n [shape=plaintext, label=""]; // dummy state
    n -> n4; // arc to the start state from nowhere
    n4 -> n0 [label="0"];
    n0 -> n1 [label="0"];
    n0 -> n2 [label="1"];
    n0 -> n3 [label="0"];
    n1 -> n1 [label="0"];
    n1 -> n2 [label="1"];
    n1 -> n3 [label="0"];
    n2 -> n1 [label="0"];
    n2 -> n2 [label="1"];
    n2 -> n3 [label="0"];
    label="NFA"
  }
```

```
  subgraph "clusterd" {
    color=blue;
    d2 [shape=doublecircle];
    d [shape=plaintext, label=""]; // dummy state
    d -> d0; // arc to the start state from nowhere
    d0 -> d1 [label="0"];
    d1 -> d2 [label="0"];
    d1 -> d3 [label="1"];
    d2 -> d2 [label="0"];
    d2 -> d3 [label="1"];
    d3 -> d2 [label="0"];
    d3 -> d3 [label="1"];
    label="DFA"
  }

  subgraph "clusterm" {
    color=blue;
    m0 [shape=doublecircle];
    m [shape=plaintext, label=""]; // dummy state
    m -> m1; // arc to the start state from nowhere
    m0 -> m0 [label="0"];
    m0 -> m2 [label="1"];
    m1 -> m2 [label="0"];
    m2 -> m0 [label="0"];
    m2 -> m2 [label="1"];
    label="min DFA"
  }
}
```

Wynik w postaci diagramów: