Metody Kompilacji
Wykład 6
Analiza Leksykalna cd

Analiza Leksykalna

Definicje Regularne (*Regular Definitions*)

- Jeśli Σ jest alfabetem symboli podstawowych, to definicją regularną jest sekwencja :

$$d_1 \rightarrow r_1$$
$$d_2 \rightarrow r_2$$
$$\ldots$$
$$d_n \rightarrow r_n$$

Analiza Leksykalna

Definicje Regularne

gdzie :

1. każde d_j jest to unikatowy symbol (d_j są różnymi symbolami), nie należący do alfabetu Σ.

2. Każde r_i jest wyrażeniem regularnym nad symbolami z alfabetu $\Sigma \cup \{d_1, d_2, \ldots, d_{i-1}\}$.

Analiza Leksykalna

Definicje Regularne

- Definicje regularne nie mogą być rekurencyjne:

digits $\rightarrow$ ~~digit digits~~ | digit błąd!

Analiza Leksykalna

Definicje Regularne

- **Przykład**: Definicja regularna dla nazw w języku C:

  letter -> A | B |...| Z | a | b |...| *z* |...

  digit -> 0| 1 |...| 9

  id -> *letter_* (*letter_* | *digit*)*,

  gdzie *letter_* oznacza dowolną literę lub znak podkreślenia.

Analiza Leksykalna

Rozszerzenia wyrażeń regularnych

- Rozszerzenia, które zostały po raz pierwszy wprowadzone do narzędzia *Lex*:

  1. Jedno lub więcej wystąpień. Jednoskładnikowy postfiksowy operator + reprezentuje domknięcie dodatnie wyrażenia regularnego i jego języka.

Analiza Leksykalna

Rozszerzenia wyrażeń regularnych

- Oznacza, że jeśli r jest wyrażeniem regularnym, to $(r)+$ oznacza język $(L(r))+$.

Analiza Leksykalna

Rozszerzenia wyrażeń regularnych

Operator "+" (domknięcie dodatnie) ma ten sam priorytet i łączność co i operator "*" (domknięcie).

Dwa użyteczne prawa algebraiczne:

$$r* = r+ \mid \varepsilon$$

$$r+ = rr* = r*r$$

Analiza Leksykalna

Rozszerzenia wyrażeń regularnych

2. Zero lub jedno wystąpienie.

Jednoskładnikowy postfiksowy operator **?** oznacza "zero lub jedno wystąpienie." To jest, *r*? jest równoważne z $r|\varepsilon$ lub innymi słowy: $L(r?) = L(r) \cup \{\varepsilon\}$.

Operator **?** ma ten sam priorytet i łączność co operatory * i +.

Analiza Leksykalna

Rozszerzenia wyrażeń regularnych

3. Wyrażenie regularne $\mathbf{a_1|a_2|...|a_n}$, gdzie $\mathbf{a_i}$ są to symbole alfabetu, może być zastąpione przez skrót $[\mathbf{a_1\ a_2}\ ...]$.

Co więcej, gdy $\mathbf{a_1, a_2, . . . , a_n}$ tworzą logiczny ciąg, np. kolejne litery duże, litery małe lub cyfry, możemy zastąpić je przez $\mathbf{a_1}$-$\mathbf{a_n}$:

$$[\mathbf{a\text{-}z}] = \mathbf{a} \mid \mathbf{b} \mid \mathbf{c} \mid \ldots \mid \mathbf{z}$$

Analiza Leksykalna

Rozszerzenia wyrażeń regularnych

Przykład: definicje regularne jak niżej

letter -> A | B |...| Z | a | b |...| z |...

digit -> 0| 1 |...| 9

id -> *letter_* (*letter_* | *digit*)*

gdzie *letter_* oznacza dowolną literę lub znak podkreślenia

możemy zapisać w postaci:

Analiza Leksykalna

Rozszerzenia wyrażeń regularnych

*letter_* -> [A- Za -z_]

digit -> [0-9]

id -> *letter_*(*letter_* | *digit*)*

Analiza Leksykalna

Rozpoznawanie leksemów

- Rozważmy następujący przykład:

stmt -> **if** *expr* **then** *stmt*

| **if** *expr* **then** *stmt* **else** *stmt*

| ε

expr -> *term* **relop** *term*

| *term*

term -> **id** | **number**

Analiza Leksykalna

Rozpoznawanie leksemów

- Wzorce leksemów:

digit -> [0-9]

digits -> *digit*+

number ->*digits* (.*digits*)? (E [+-]? *digits*)?

Letter-> [A-Za-z]

id -> *letter* (*letter* | *digit*)*

if -> if *then* -> then *else* -> else

relop -> < | > | < = | > = | = | < >

Analiza Leksykalna

- **Rozpoznawanie leksemów**
- Ponadto, możemy przypisać analizatorowi leksykalnemu zadanie rozpoznawania znaków białych określonych przez definicję regularną:

  ws -> (**blank | tab | newline**)+

  gdzie **blank, tab**, i **newline** są to symbole abstrakcyjne, których używamy do wyrażania znaków ASCII o tych samych nazwach.

Analiza Leksykalna

Rozpoznawanie leksemów

- Token *ws* różni się od innych żetonów tym, że gdy zostaje rozpoznany nie jest przekazywany do parsera, zamiast tego analizator leksykalny przechodzi do rozpoznawania następnego leksemu, który następuje po znaku białym.

Analiza Leksykalna

Leksemy, tokeny i atrybuty

Wyrażenia regularne	Token	Wartość atrybutu
białe znaki	-	-
if, then, else	**if, then, else**	-
id, liczba	**Id, number**	Wskaźnik do tablicy symboli
<, <=,>,>=,…	**relop**	LT, LE, GT, GE,….

Analiza Leksykalna

Diagramy Przejść (Transition Diagrams)

- Jako krok pośredni w budowie analizatora leksykalnego, będziemy najpierw konstruować schematy blokowe, nazywane *diagramami przejść*.
- Diagramy przejść mają zbiór węzłów, rysowane jako okręgi, i są nazywane stanami.

Analiza Leksykalna

Diagramy Przejść

- Każdy stan reprezentuje warunek, który może wystąpić podczas procesu skanowania wejścia celem rozpoznawania leksemu, który pasuje do jednego z kilku wzorców.

Analiza Leksykalna

Diagramy Przejść

- Krawędzie są kierowane z jednego stanu do jakiegoś innego.
- Każda krawędź jest oznaczona przez symbol lub zbiór symboli.
- Jeśli jesteśmy w stanie *s* i następnym symbolem jest symbol *a*, to szukamy krawędzi wychodzącej ze stanu *s* i oznakowanej przez symbol *a*.

Analiza Leksykalna

Diagramy Przejść

- Jeśli znajdziemy taką krawędź, to przechodzimy do stanu, do którego prowadzi ta krawędź.

  Będziemy zakładać, że wszystkie nasze diagramy przejść są deterministyczne, co oznacza, że taka sama etykieta nie może oznaczać dwóch lub więcej krawędzi wychodzących z tego samego stanu.

Analiza Leksykalna

Diagramy Przejść

- Ważne pojęcia związane z diagramami przejść:
- 1. Stany końcowe: Stany te wskazują, że leksem został rozpoznany, one są oznaczone na rysunku okręgiem podwójnym.

Analiza Leksykalna

Diagramy Przejść

Jeśli ma być wykonana jakaś akcja (zazwyczaj zwrot tokena do parsera), to dołączamy tę akcję do stanu akceptującego.

Analiza Leksykalna

Diagramy Przejść

- 2. Ponadto, jeśli jest to konieczne, aby przesunąć wskaźnik symbolu o jedną pozycję do przodu (czyli leksem nie zawiera symbolu, który prowadzi do stanu akceptującego), wówczas należy dodatkowo wstawić znak "*" obok stanu końcowego.

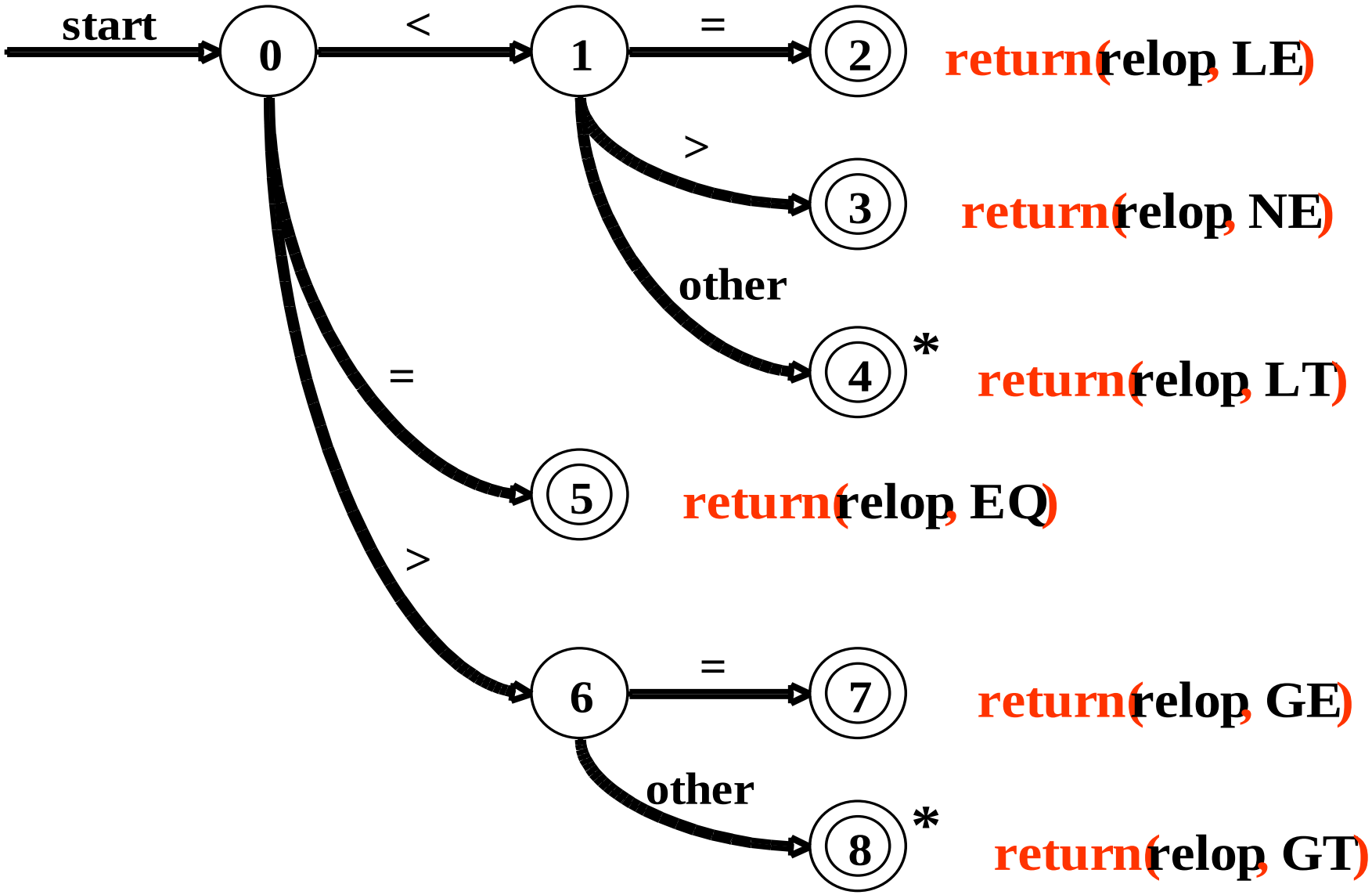
Analiza Leksykalna

Diagramy Przejść

- 3. Jeden ze stanów jest nazywany stanem początkowym i oznaczony jest krawędzią wchodzącą o nazwie *start*.

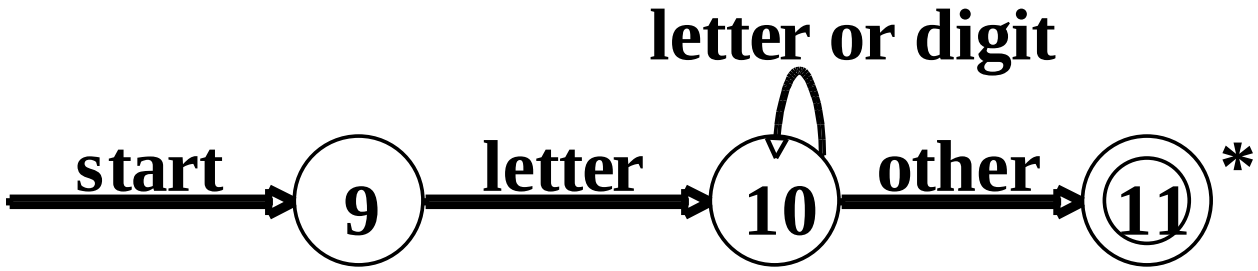
Jest to stan diagramu, od którego zaczynamy rozpoznawanie leksemu.

Diagram przejść dla relop



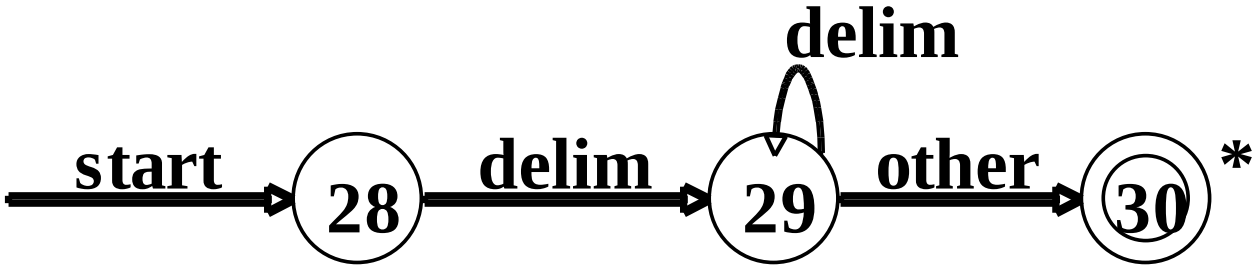
Rozpoznawanie nazw i znaków białych

id :



Delimiter – ogranicznik,
separator – białe znaki,
komentarze.

delim:



Analiza Leksykalna

Implementacja analizatora leksykalnego

- Możemy wprowadzić zmienną o nazwie *state* do przechowywania bieżącego stanu diagramu przejść.
- Wtedy możemy zaimplementować analizator leksykalny w oparciu o konstrukcję *switch*.

Analiza Leksykalna

Implementacja diagramu przejść

Każdy stan zawiera odpowiedni segment kodu.

Jeśli są krawędzie wychodzące ze stanu, to jego kod odczytuje jeden znak i wybiera jedną krawędź wychodzącą, jeśli taka istnieje.

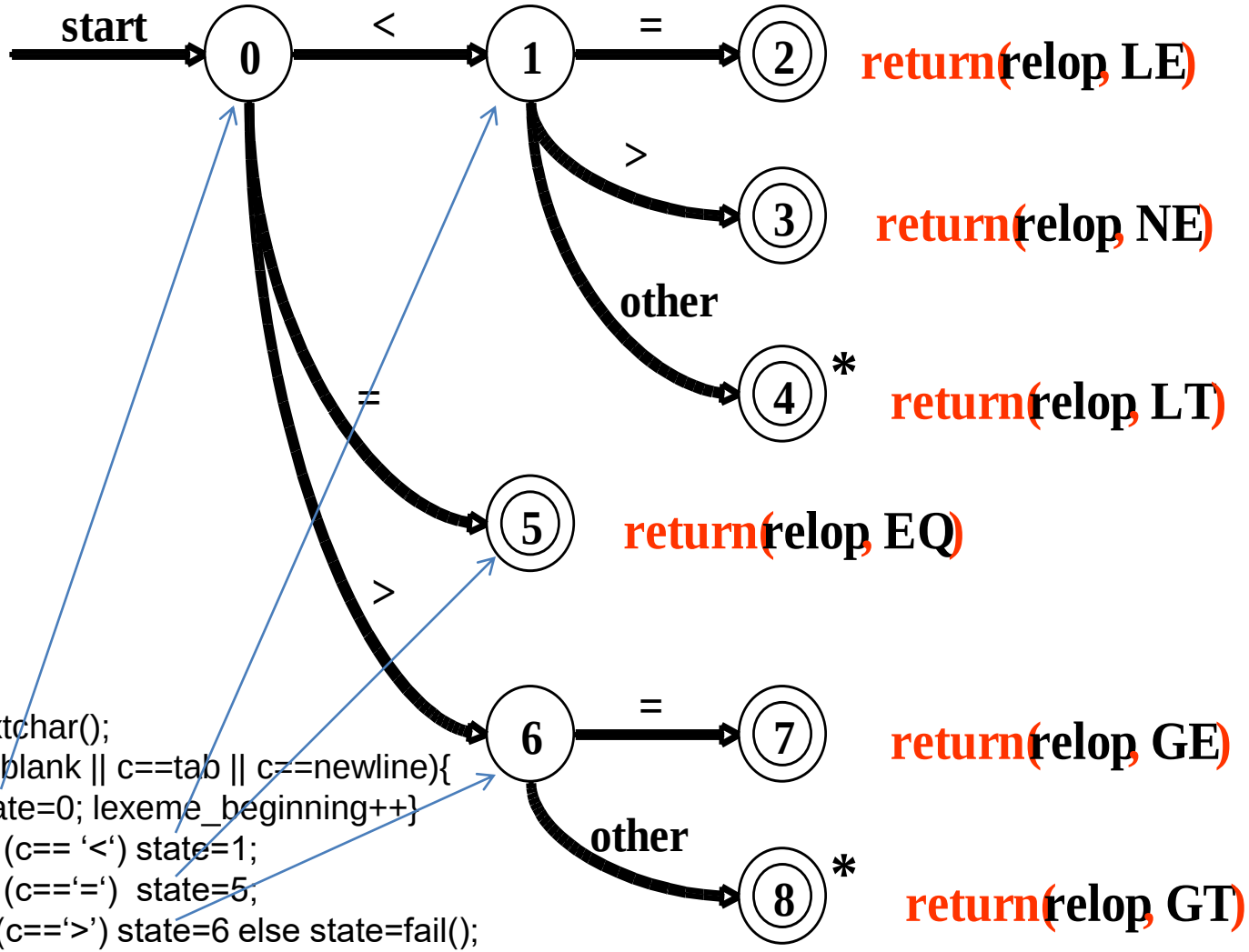
Używa funkcji *nextchar*(), aby wczytać następny znak z bufora wejściowego.

Analiza Leksykalna

Implementacja analizatora leksykalnego

```
while (1) {
 switch(state) {
   case 0: c=nextchar();
           if (c==blank || c==tab || c==newline){
                 state=0; lexeme_beginning++}
           else if (c== ‘<‘) state=1;
           else if (c==‘=‘)  state=5;
           else if(c==‘>’) state=6 else state=fail();
            break
    case 9: c=nextchar();
           if (isletter(c)) state=10;
           else state=fail(); break
 … }}}
```

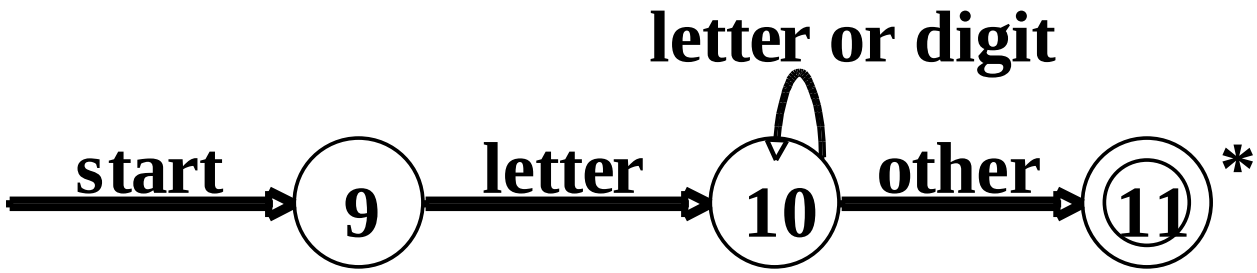
Diagram przejść dla relop




```
while (1) {
    switch(state) {
      case 0: c=nextchar();
              if (c==blank || c==tab || c==newline){
                   state=0; lexeme_beginning++}
              else if (c== '<') state=1;
              else if (c=='=')  state=5;
              else if(c=='>') state=6 else state=fail();
               break
```

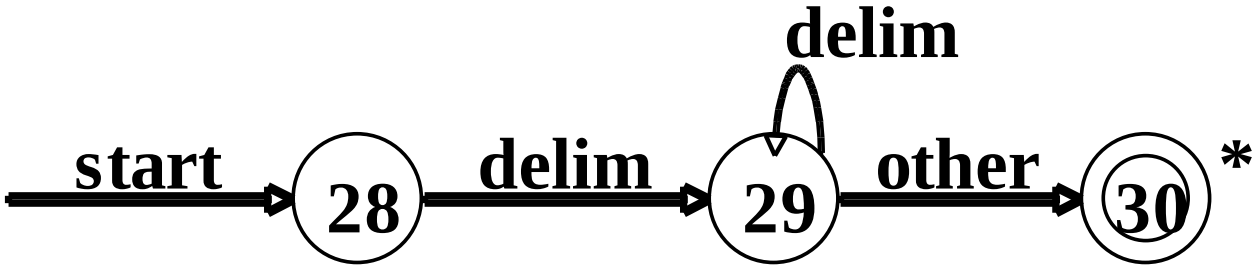
Rozpoznawanie nazw i znaków białych

id :




```
case 9: c=nextchar();
                if (isletter( c)) state=10;
                else state=fail(); break
```

delim:



Narzędzie Lex

Książki

- lex & yacc, 2nd Edition
  - John R.Levine, Tony Mason & Doug Brown
  - O'Reilly
  - ISBN: 1-56592-000-7

- Mastering Regular Expressions
  - Jeffrey E.F. Friedl
  - O'Reilly
  - ISBN: 1-56592-257-3

Analiza Leksykalna

Narzędzie do analizy leksykalnej *Lex*

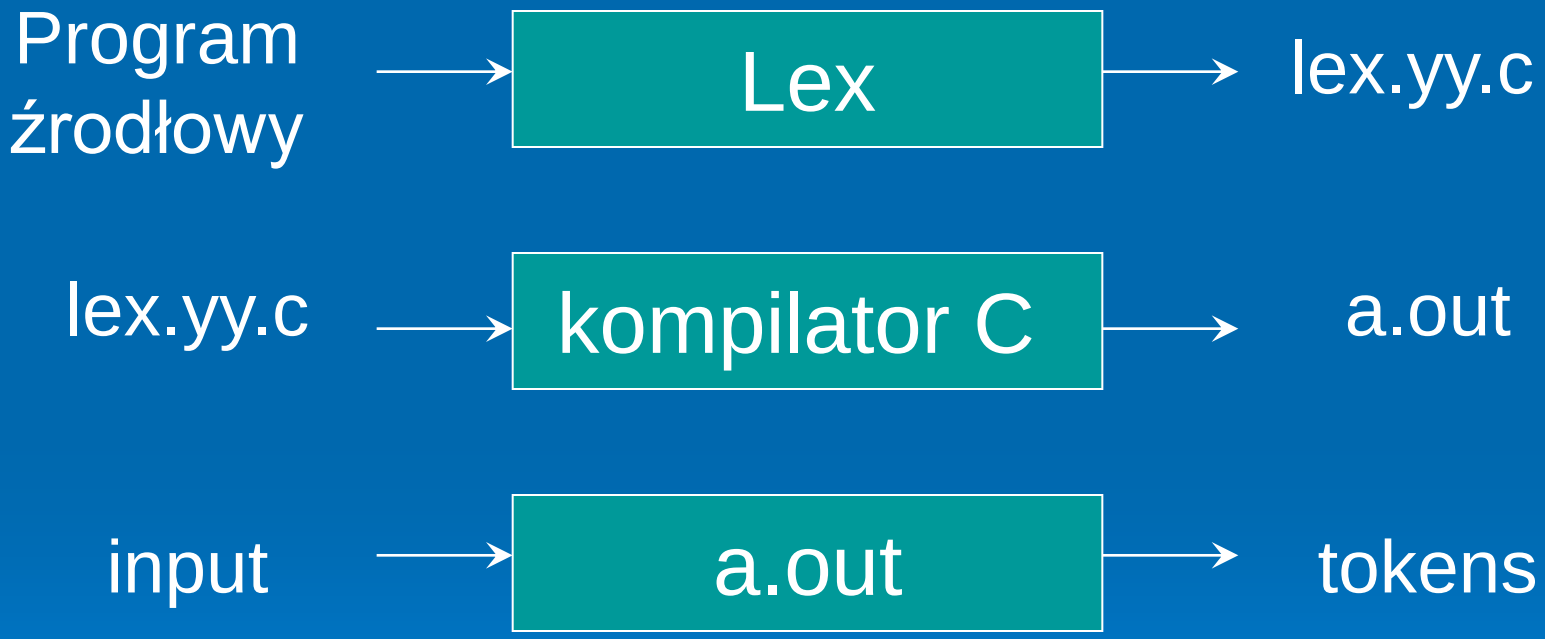
- Program wejściowy dla narzędzia *Lex* tworzymy w języku *Lex*, natomiast samo narzędzie nazywamy kompilatorem *Lex*.

Analiza Leksykalna

Narzędzie do analizy leksykalnej *Lex*

Kompilator *Lex* konwertuje tokeny wejściowe na diagram przejść oraz generuje kod symulujący diagram przejść, który jest zapisywany do pliku o nazwie lex.yy.c.

Narzędzie do analizy leksykalnej *Lex*



Analiza Leksykalna

Struktura programu w języku *Lex*

deklaracje

%%

reguły translacji

%%

funkcje pomocnicze

Analiza Leksykalna

Struktura programu w języku *Lex*

- Sekcja deklaracji zawiera deklaracje zmiennych, identyfikatory zadeklarowane jako stałe, np. nazwę tokena oraz definicje regularne.

Analiza Leksykalna

Struktura programu w języku *Lex*

- Reguły translacji mają postać:

  Wzorzec(Pattern) { Akcja(Action) }

Analiza Leksykalna

Struktura programu w języku *Lex*

- Każdy wzorzec jest wyrażeniem regularnym, które może korzystać z definicji regularnych określonych w sekcji deklaracji.

Analiza Leksykalna

Struktura programu w języku *Lex*

- Akcje są fragmentami kodu, zwykle napisane w języku C, choć modyfikacje narzędzia *Lex* używają innych języków.

Analiza Leksykalna

Struktura programu w języku *Lex*

- Trzecia sekcja zawiera dodatkowe funkcje, które są stosowane w akcjach.

- Alternatywnie, funkcje te mogą być kompilowane osobno i ładowane za pomocą analizatora leksykalnego.

Analiza Leksykalna

Struktura programu w języku *Lex*

- Wywołany przez parser, analizator leksykalny zaczyna czytać znak po znaku program źródłowy dopóki nie znajdzie najdłuższego prefiksu, który pasuje do jakiegoś wzorca. Następnie wykonuje działania odpowiedniej akcji.

Analiza Leksykalna

Struktura programu w języku *Lex*

Analizator leksykalny zwraca jedną wartość, nazwę symboliczną, do parsera, ale może w razie potrzeby korzystać ze zmiennej dzielonej o nazwie `yylval` celem przekazania dodatkowej informacje o znalezionym leksemie.

Przykład

```
%{                                  / / Deklaracje pomocnicze  (w C)
#define LT   24
#define LE   25
#define EQ 26
 ...
%}                                  // definicje regularne
delim          [ \t\n]
ws             {delim}+
letter         [A-Za-z]
digit          [0-9]
id             {letter}({letter}|{digit})*
.....
%%
```

Przykład

// akcje w C

```
{ws}        { /* brak akcji*/ }
if          {return (IF);}
then        {return (THEN);}
else        {return (ELSE);}
{id}        {yylval=install_id(); return (ID);}
{number}    {yylval=install_num(); return (NUMBER);}
"<"         {yylval=LT; return (RELOP);}
"<="        {yylval=LE; return (RELOP);}
 ...
%%                                  // procedury w C
install_id() { …   }
install_num() { … }
```

Analiza Leksykalna

Rozwiązywanie konfliktów w *Lex*

- *Lex* stosuje dwie zasady w celu wybrania właściwego leksemu, gdy kilka prefiksów pasuje do jednego lub więcej wzorców:

  1. Wybiera najdłuższy prefiks.

  2. Jeśli najdłuższy prefiks pasuje do dwóch lub więcej wzorców, wybiera wzorzec, który jest wcześniejszy w tekście programu *Lex*.

Analiza Leksykalna

- **Rozwiązywanie konfliktów w Lex**
- **Przykład**: Pierwsza zasada mówi nam żeby czytać dalej litery i cyfry, aby znaleźć najdłuższy prefiks grupy znaków pasujących do identyfikatora.
- Mówi nam także żeby potraktować ciąg znaków „<=„ jako jeden leksem.

Analiza Leksykalna

- **Rozwiązywanie konfliktów w Lex**

- Druga zasada sprawia, że rozpoznane słowo jest traktowane jako słowo kluczowe, jeżeli deklarujemy listę słów kluczowych przed deklaracją identyfikatora w programie *Lex*.

Analiza Leksykalna

- Na przykład, jeśli słowo `then` ma najdłuższy prefiks, który pasuje do wzorca oraz `then` występuje wcześniej niż {id}, to jest zwracany token THEN (nie ID) .

Analiza Leksykalna

Automaty skończone

- Zajmiemy się teraz pytaniem co robi *Lex* z programem wejściowym?

Analiza Leksykalna

Automaty skończone

- W oparciu o program wejściowy, *Lex* tworzy automat skończony (*finite automata*).
- Automat skończony jest podobny do diagramu przejść z kilkoma różnicami:

Analiza Leksykalna

- **Automaty skończone**

- 1. Automaty skończone służą tylko do rozpoznawania: po prostu zwracają odpowiedź "tak(rozpoznany)" lub "nie(nierozpoznany)" w stosunku do ciągu wejściowego.

Analiza Leksykalna

- **Automaty skończone**

  2. Automaty skończone dzielimy na

  (a) niedeterministyczne (*Nondeterministic finite automata* - NFA), które nie mają ograniczeń na etykiety krawędzi:

  tym samym symbolem można oznaczyć kilka krawędzi wychodzących z tego samego stanu oraz napis pusty może być etykietą krawędzi.

Analiza Leksykalna

- **Automaty skończone**

  (b) deterministyczne (*Deterministic finite automata* -DFA): każda krawędź, wychodząca z jakiegoś stanu, posiada unikatową etykietę.

Analiza Leksykalna

- **Automaty skończone**
- Zarówno deterministyczne automaty skończone jak i niedeterministiczne są w stanie rozpoznać te same języki oparte na wyrażeniach regularnych.

Analiza Leksykalna

- **Niedeterministyczne automaty skończone**
- Niedeterministyczny automat skończony zawiera:

  1. Skończony zbiór stanów S.

  2. Alfabet wejściowy Σ. Zakładamy, że napis pusty ε nie należy do alfabetu Σ.

Analiza Leksykalna

- **Niedeterministyczne automaty skończone**

  3. Funkcję przejścia, która określa dla każdego stanu i dla każdego symbolu należącego do zbioru $\Sigma \cup \{\varepsilon\}$ zbór kolejnych stanów, do których można przejść.

Analiza Leksykalna

- **Niedeterministyczne automaty skończone**

4. Jeden stan początkowy(startowy) *s0.*

5. Zbiór stanów końcowych(akceptowalnych) *F*, który jest podzbiorem zbioru *S.*

Analiza Leksykalna

- **Niedeterministyczne automaty skończone**
- Możemy reprezentować zarówno NFA jak i DFA przez graf przejść, w którym węzły reprezentują stany, natomiast krawędzie oznakowane reprezentują funkcję przejścia.

Analiza Leksykalna

- **Niedeterministyczne automaty skończone**
- Istnieje krawędź oznakowana etykietą *a* ze stanu *s* do stanu *t* wtedy i tylko wtedy, gdy *t* jest jednym z następnych stanów dla stanu *s* i wejścia *a*.

Analiza Leksykalna

- **Niedeterministyczne automaty skończone**
- Graf ten jest bardzo podobny do diagramu przejść, z następującymi wyjątkami:

  a) ten sam symbol może oznaczyć kilka krawędzi wychodzących z jednego stanu do kilku różnych stanów;

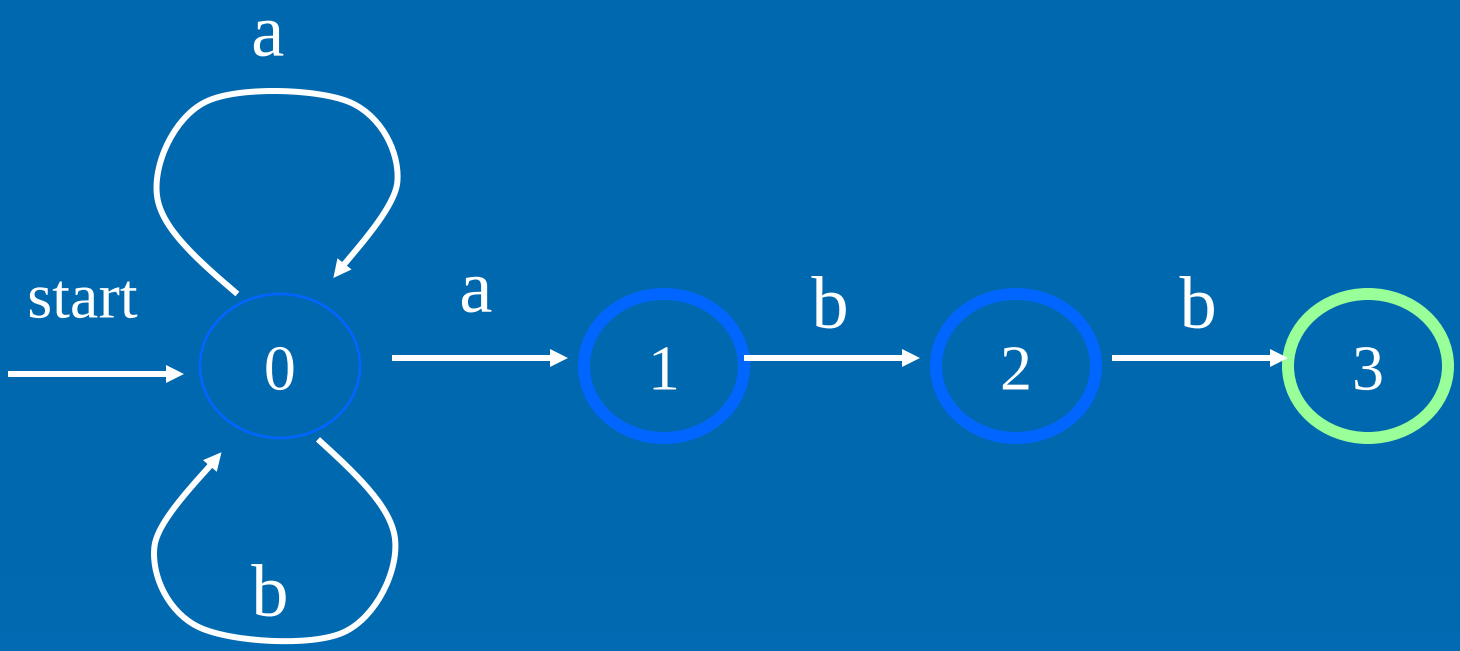
Analiza Leksykalna

- **Niedeterministyczne automaty skończone**

  b) krawędź może być oznakowana za pomocą napisu pustego, który zaznaczamy jako ε.

Przykład NFA

rozpoznawanie wyrażenia regularnego (a|b)*abb



Zbiór stanów= {0,1,2,3}

Język = {a,b}

Stan startowy S0, stan końcowy S3

Analiza Leksykalna

- **Tablice przejść**
- Możemy również reprezentować NFA przez tablicę przejść, której wiersze odpowiadają stanom, natomiast kolumny odpowiadają symbolom wejściowym i ε.
- Wpis dla danego stanu i wejścia jest to wartość zwracana przez funkcję przejścia dla tych argumentów.

Analiza Leksykalna

- **Tablice przejść**

- Jeśli jakiś element tablicy zawiera symbol ∅, oznacza to, że taki element nie zawiera żadnej informacji.

Funkcja przejścia

Funkcja przejść zaimplementowana w postaci tablicy:

Stan	Wejście	
	a	b
0	{0,1}	{0}
1	--	{2}
2	--	{3}

Analiza Leksykalna

- **Rozpoznawanie leksemów**

- NFA akceptuje ciąg znaków *x*, wtedy i tylko wtedy, gdy istnieje jakaś ścieżka w grafie przejść od stanu początkowego do jednego z stanów końcowych, takiego, że symbole wzdłuż ścieżki tworzą *x*.

Analiza Leksykalna

- **Rozpoznawanie leksemów**

- Należy pamiętać, że etykiety oznakowane przez ε są ignorowane.

Analiza Leksykalna

- **Deterministyczne automaty skończone**

- Jeśli używamy tablicy przejścia do reprezentowania DFA, to każdy pojedynczy wpis reprezentuje pojedynczy stan, dlatego zapisujemy ten stan bez nawiasów.

Analiza Leksykalna

- **Deterministyczne automaty skończone**
- DFA pozwala na bardzo łatwe rozpoznawanie leksemów.

- Każde wyrażenie regularne i każdy NFA mogą być konwertowane do DFA akceptującego ten sam język.

Analiza Leksykalna

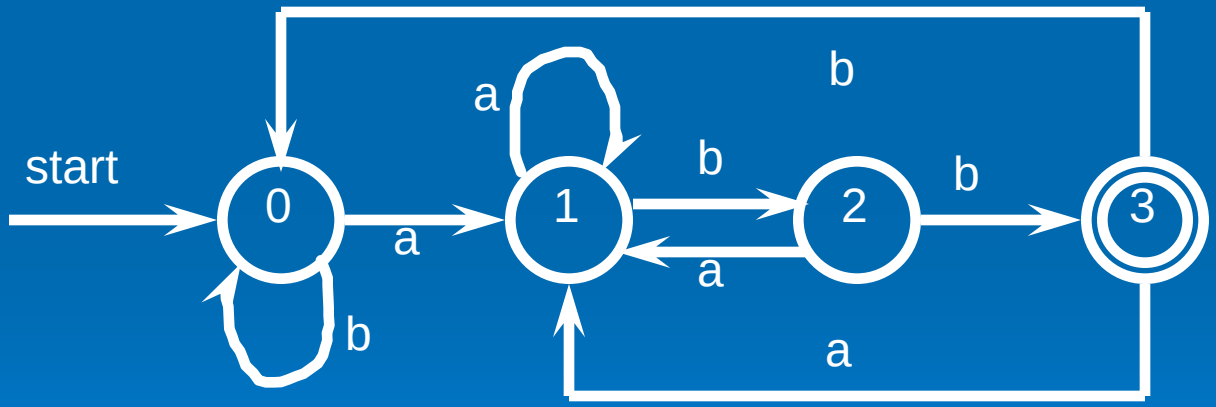
- **Algorytm: Symulacja DFA**
- Wejście: Napis *x*, który kończy się zna*kiem* **eof**. DFA *D* ze stanem startowym *s0* i stanami końcowymi *F* oraz funkcja przejść *move*.
- Wyjście : Odpowiedź „tak(yes)" jeśli *D* rozpoznaje *x* inaczej „nie(no)„ .

Symulacja DFA

```
begin
   s := s0;
   c := nextchar;
   while c <> eof do begin
      s := move(s, c);    // funkcja przejść
      c := nextchar
   end;
   if s is in F then return "yes"
   else return "no"
end.
```

Przykład

$(a \mid b)^*abb$

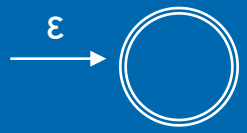


Konwersja RE do NFA

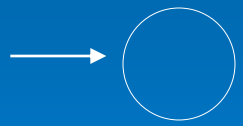
Konwersje bazowe:

R=a

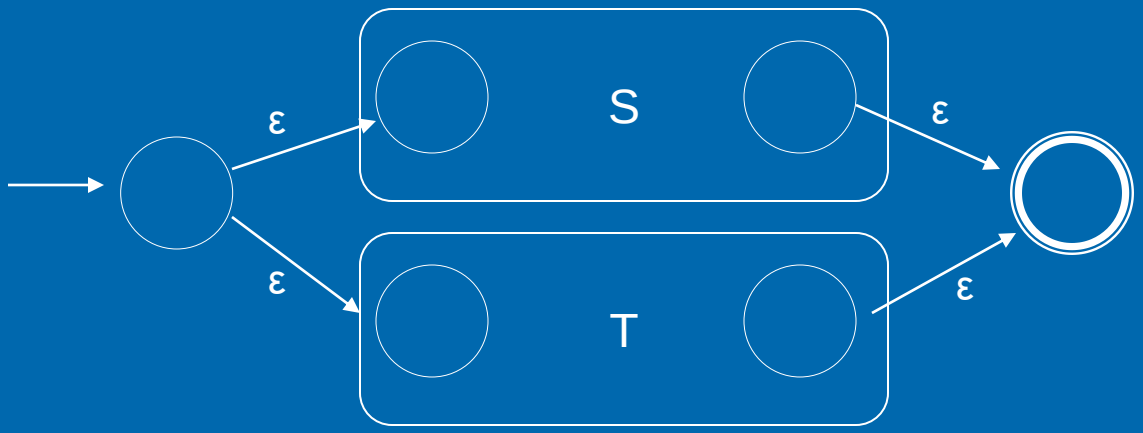
R=ε



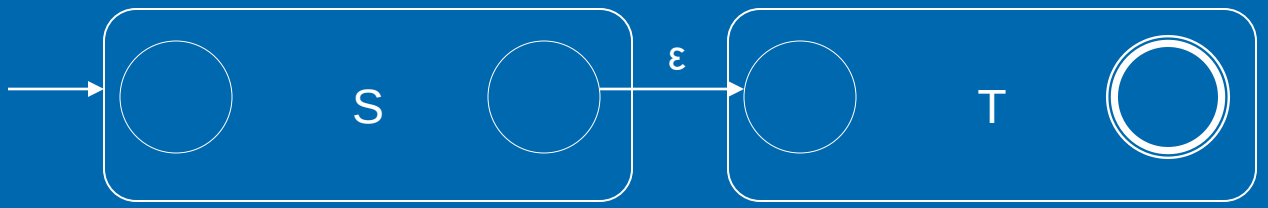
R=Ø



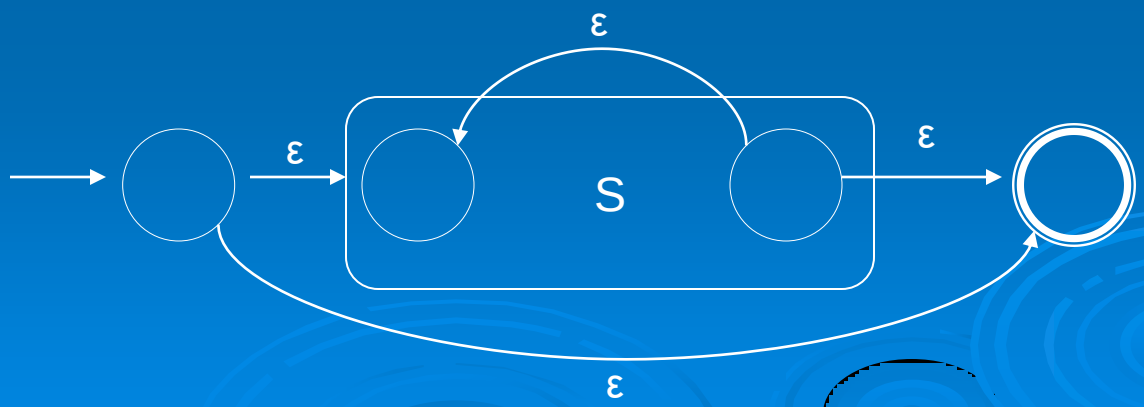
R=S ∪ T



R=ST



R=S*

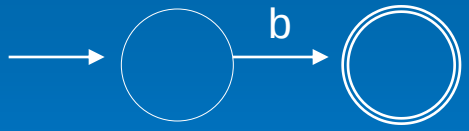


Konwersja RE do NFA

- Zamień R= (ab+a)* na NFA
  - Zaczynamy od najprostszych elementów

a

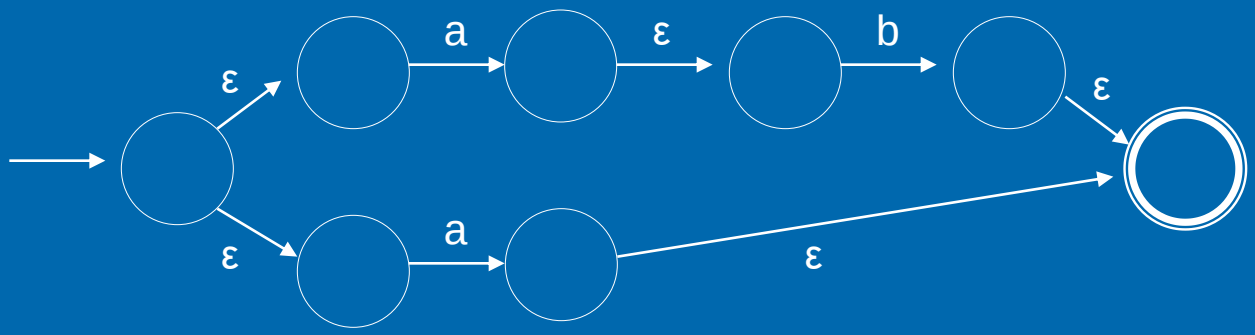
b



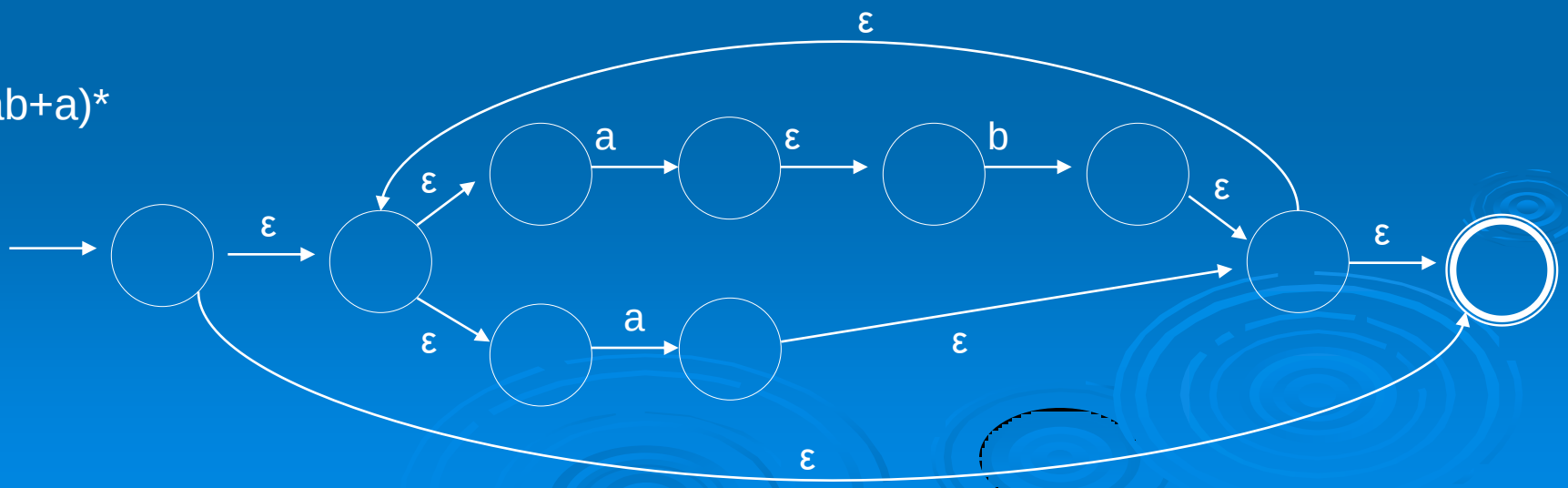
ab

Konwersja RE do N

ab+a



(ab+a)*



Dziękuję za uwagę