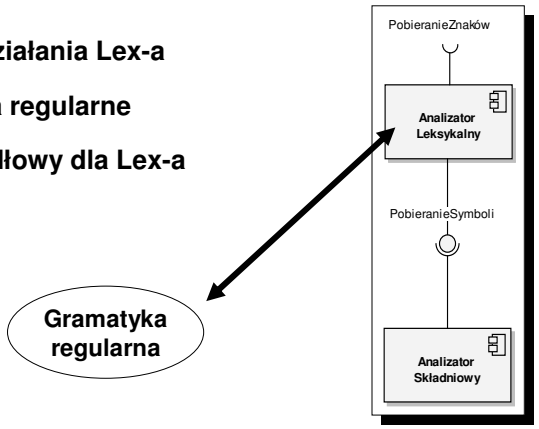
Analiza leksykalna - Lex

- **Sposób działania Lex-a**
- **Wyrażenia regularne**
- **Tekst źródłowy dla Lex-a**



Języki formalne i kompilatory, © by Michał Śmiałek

Co to jest Lex?

Lex jest generatorem programów służących do analizy leksykalnej strumieni wejściowych.

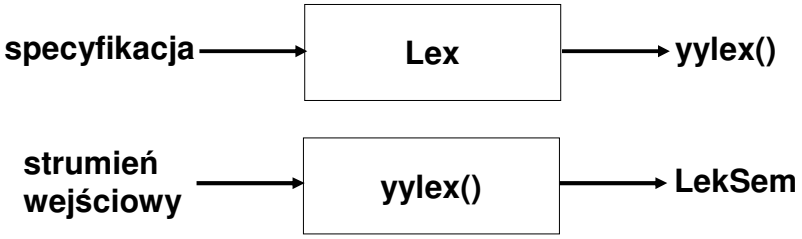
- Lex przyjmuje na wejściu opis w formie serii wyrażeń regularnych;
- Lex produkuje program w języku C, który z wyrażeń regularnych tworzy automat skończony;
- Automat wyprodukowany przez Lex-a pozwala wykrywać w ciągu wejściowym sekwencje znaków zgodne z wzorcami określonymi przez zadane wyrażenia regularne;
- Wygenerowany program może podejmować zadane przez nas działania po wykryciu stanu końcowego automatu (zaakceptowaniu określonego ciągu znaków);

Program wyprodukowany przez Lex-a można włączyć do swojego programu jako moduł odpowiedzialny za wstępną analizę leksykalną.

Języki formalne i kompilatory © by Michał Śmiałek

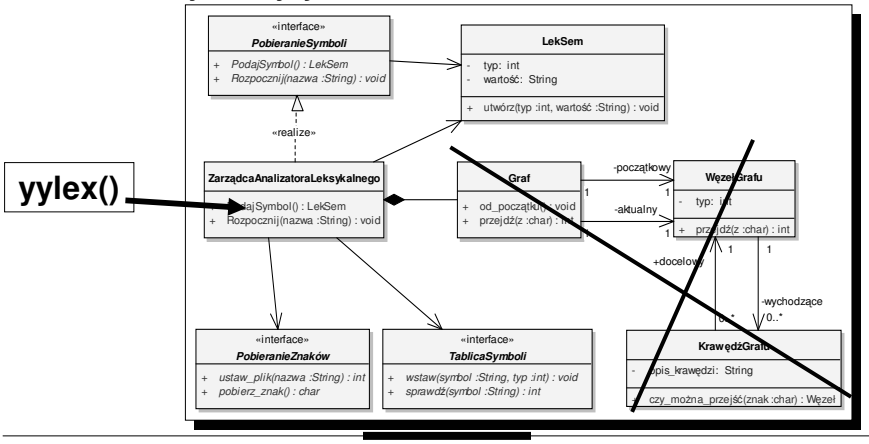
Jak działa Lex?

Program Lex przyjmuje specyfikację w odpowiednim formacie. Wyjściem jest procedura „yylex" umożliwiająca przetwarzanie strumienia wejściowego w odpowiednio przetworzony strumień wyjściowy (ciąg leksemów).



Lex a struktura kompilatora

Funkcja wygenerowana przez program Lex (yylex) zastępuje klasy stanowiące strukturę automatu skończonego. Można ją zatem użyć zamiast funkcji PodajSymbol!



Wyrażenia regularne

Wyrażenie regularne (ang. regular expression), to wyrażenie określające sposób wyszukiwania lub zastępowania napisów, zawierające specjalne znaki i konteksty dopasowujące ciągi znaków w przetwarzanym tekście.

Wyrażenie regularne to swego rodzaju wzorzec, reguła opisująca pewien uogólniony napis (ciąg znaków). Jeżeli znaki tworzące dany napis spełniają reguły danego wyrażenia regularnego, to mówimy o tym wyrażeniu, że *pasuje* do danego napisu.

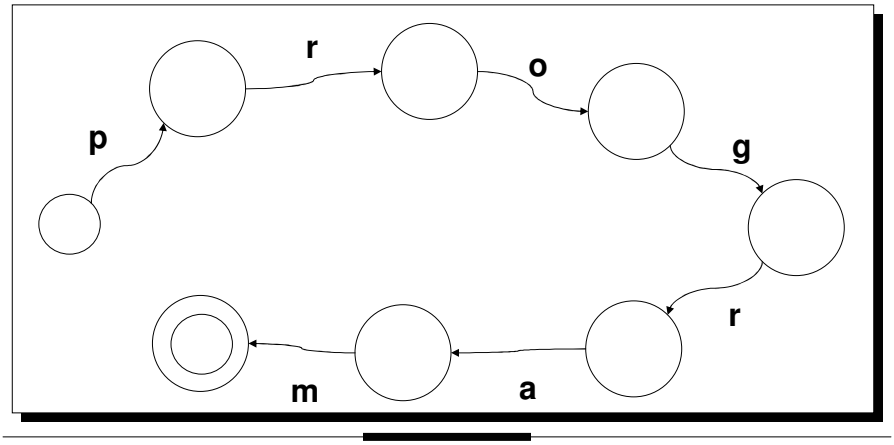
Każde wyrażenie regularne można zamienić na odpowiadający mu automat skończony (i odwrotnie).

Następne slajdy zawierają przykłady wyrażeń regularnych (w składni programu Lex) i odpowiadających im automatów.

Pojedyncze znaki

Wyrażenie regularne: program

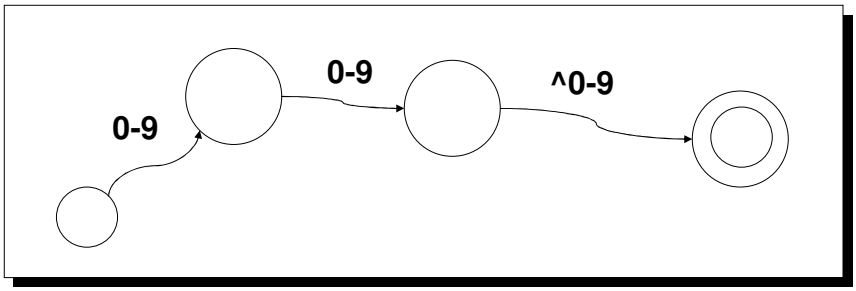
Akceptowane ciągi znaków: program



Zbiory znaków

Wyrażenie regularne: [0123456789][0-9][^0-9]

Akceptowane ciągi znaków: 27a, 31%, 99a, 01c, 75g, 56t ...

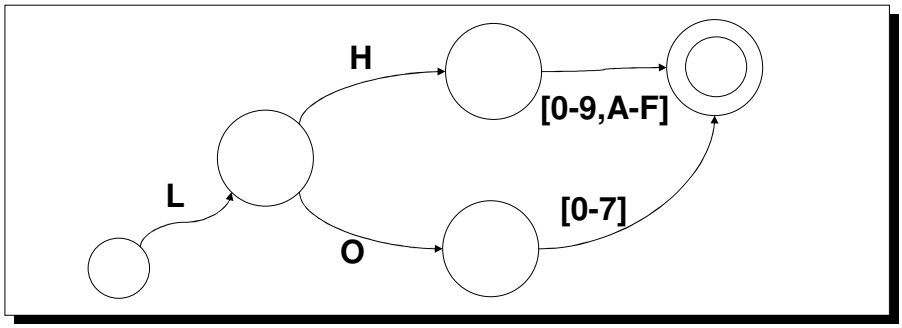


Uwaga: znak specjalny ^ oznacza „wszystko prócz”

Alternatywne wyrażenia

Wyrażenie regularne: L(H[0-9,A-F]|O[0-7])

Akceptowane ciągi znaków: LH0, LH9, LO7, LO1, ...

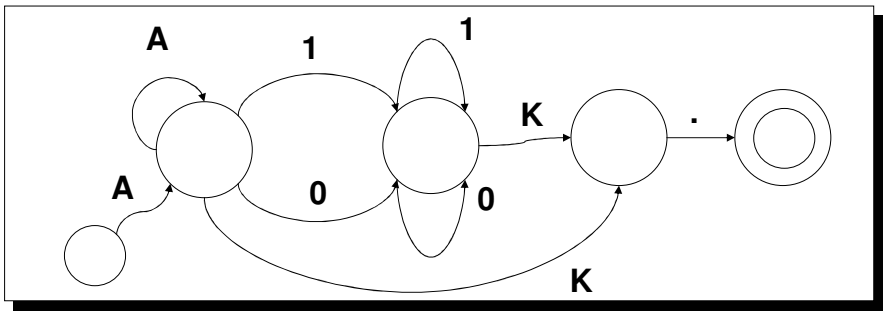


Uwaga: wyrażenia alternatywne najczęściej umieszczamy w nawiasach „()” (grupowanie)

Dowolne znaki i dowolnie długie ciągi znaków

Wyrażenie regularne: A+(0|1)*K.

Akceptowane ciągi znaków: A11K%, AA001011Kf, A00011111111Ku, AKh, ...

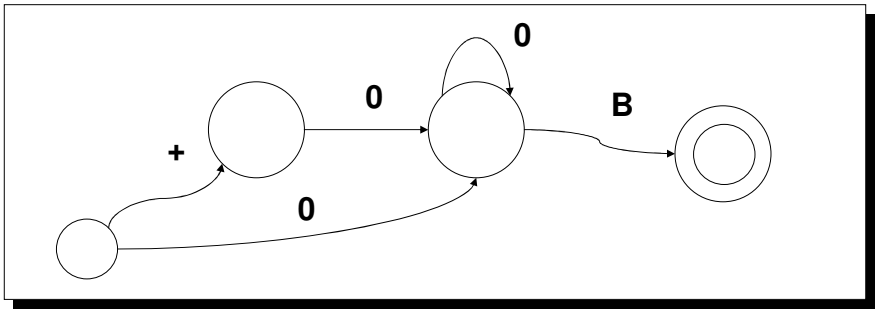


Uwaga: znak „.” oznacza dowolny znak widzialny (np. oprócz znaku nowego wiersza)

Ciągi alternatywne

Wyrażenie regularne: "+"?0+B

Akceptowane ciągi znaków: 00B, +00B, 0B, +000B, +000000B, ...



Wyrażenia regularne a gramatyki regularne

Każde wyrazenie regularne można zastąpić automatem skończonym, a każdy automat skończony posiada odpowiadającą mu gramatykę regularną.

Wyrażenie regularne z poprzedniego slajdu odpowiada nastepującej gramatyce:

Σ = {+, 0, B} Γ = {S, X}

P = {S →+X,

S →X,

X → 0X,

X → 0B}

Schemat specyfikacji dla Lex-a

Specyfikacja (plik wejściowy) dla programu Lex ma nastepujący schemat ogólny:

```
definicje pomocnicze
%%
reguły przetwarzania
%%
podprogramy pomocnicze
```

Definicje pomocnicze – umożliwiają nadawanie prostych nazw dla złożonych wyrażeń regularnych używanych wielokrotnie

Reguły przetwarzania – przyporzadkowują wyrażeniom regularnym operacje w języku C

Podprogramy pomocnicze – kod funkcji wykorzystywanych w regułach przetwarzania (i nie tylko)

Definicje pomocnicze

Format definicji pomocniczej to:

```
nazwa tłumaczenie
```

„nazwa" stanowi identyfikator

„tłumaczenie" to wyrażenie regularne, które jest zastępowane przez „nazwę"

Przykłady:

```
cyfra       [0-9]

litera      [a-zA-Z]
```

„nazwy" można stosować w regułach przetwarzania w celu skrócenia zapisu

Reguły przetwarzania

Format reguły przetwarzania to:

```
wyrażenieregularne      akcja
```

gdzie akcja to fragment programu w języku C. Najprostsza „akcja" to „;" (instrukcja pusta), co oznacza zignorowanie zaakceptowanych Operacja „ECHO;" powoduje kopiowanie znaków na standardowe wyjście. Akcje złożone z większej liczby instrukcji umieszczamy w nawiasach „{}".

Przykład:

```
[a-z]+      {slowa++; printf("%d",yyleng);}
```

Po wykryciu identyfikatora składającego się z małych liter, zwiększany jest licznik słów i wyprowadzana na standardowe wyjście długość wykrytego identyfikatora.

Przykładowa specyfikacja

```
n       [0-9]
%%
{n}+    { nowy_leksem.utwórz(1,yytext); }
[A-Za-z][A-Za-z0-9]+
        { int ident;
          ident = symbole.sprawdź(yytext);
          if (ident==0) ident = symbole.wstaw(yytext);
          if (ident<128) nowy_leksem.utwórz(ident,"");
          else nowy_leksem.utwórz(2,itoa(ident));
        }
%%
return nowy_leksem;
```

1=liczba całkowita

0 – brak symbolu w tablicy

<128 – słowa kluczowe

2 – identyfikator, nie będący słowem kluczowym