WI

Metody kompilacji

Wykład 3, 4, 5 - Analiza leksykalna

Włodzimierz Bielecki, Piotr Błaszyński

Wydział Informatyki Zachodniopomorskiego Uniwersytetu Technologicznego

27 marca 2020

Analizator leksykalny

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

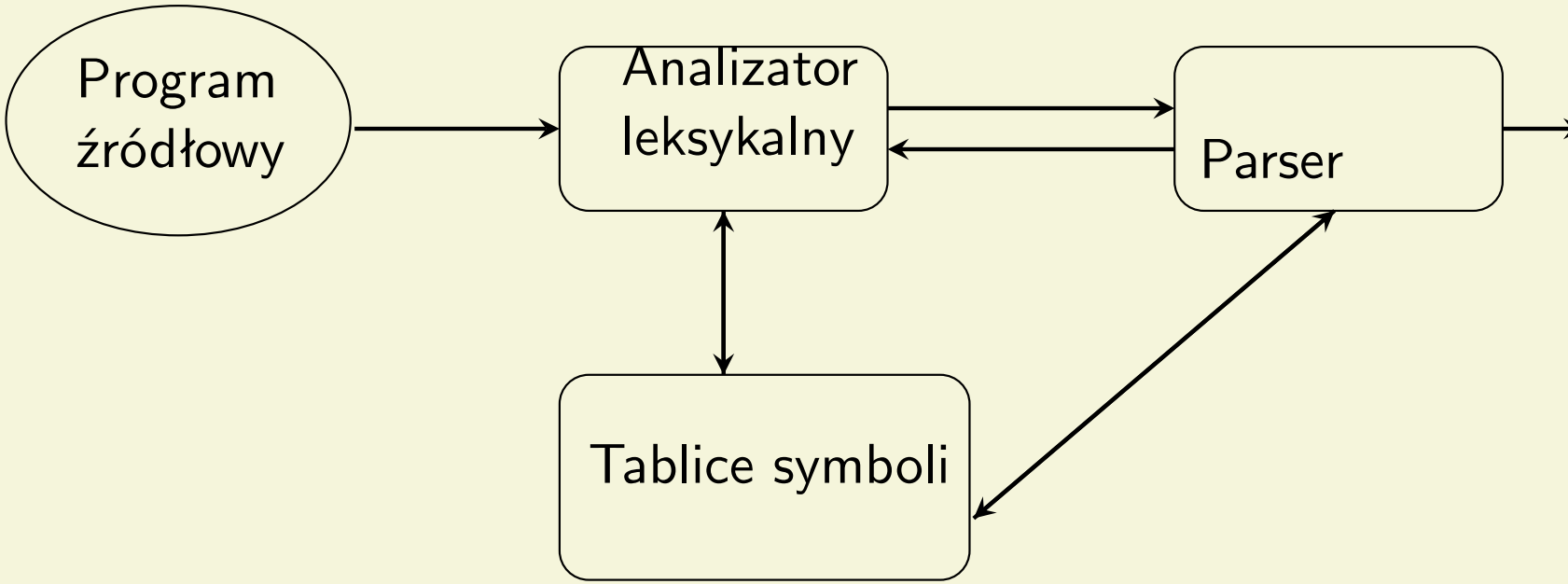
Analizator leksykalny odczytuje znaki z wejścia, rozpoznaje leksemy i produkuje tokeny. Wraz z symbolem terminalnym, który jest używany przez parser, token zawiera dodatkowe informacje w postaci wartości atrybutów. Token jest to terminal wraz z dodatkową informacją. Rola analizatora leksykalnego w procesie kompilacji jest pokazana graficznie na rysunku.

Rola analizatora leksykalnego

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA



Rysunek: Rola analizatora leksykalnego

Rola analizatora leksykalnego

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Sekwencja znaków wejściowych, która zawiera pojedynczy token, nazywa się leksemem. Można zatem powiedzieć, że analizator leksykalny izoluje parser od reprezentacji znakowej symbolu, co oznacza, że parser dostaje tokeny, a nie znaki. Głównym zadaniem analizatora leksykalnego jest wczytywanie znaków z programu źródłowego, rozpoznawanie leksemów i produkowanie tokenów (znaczników) dla każdego leksemu. Inne zadania to eliminowanie komentarzy i „znaków białych”: spacja, znak nowej linii, znak tabulacji. Kolejnym zadaniem jest współudział w obsłudze błędów generowanych przez kompilator. Na przykład analizator leksykalny może śledzić liczbę linijek kodu, liczbę znaków każdej linijki i przekazywać te dane do kompilatora.

Dlaczego analizator leksykalny jest tworzony osobno?

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Dlaczego analizator leksykalny jest tworzony osobno?

1. Prostota projektowania analizatora leksykalnego (w stosunku do projektowania analizatora syntaktycznego) jest najważniejszym czynnikiem.
2. Zwiększona jest wydajność kompilatora. Opracowane są wyspecjalizowane bardzo wydajne techniki analizy leksykalnej. Ponadto techniki buforowania do wczytywania znaków wejściowych mogą znacznie przyspieszyć kompilator.
3. Kompilator ma zwiększoną przenośność, czyli może być stosowany na różnych platformach. Specyficzne dla urządzeń wejścia osobliwości mogą być uwzględnione tylko w analizatorze leksykalnym; pozostałe części kompilatora zostają bez zmian.

Tokeny, wzorce, leksemy

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Tokeny, wzorce, leksemy. Wzorzec jest opisem postaci, którą leksem może przyjąć. W przypadku słów kluczowych wzorzec jest po prostu ciągiem znaków, które tworzą słowa kluczowe. Leksem jest ciągiem znaków w programie źródłowym, który pasuje do jakiegoś wzorca. Jest to niepodzielny element programu, dlatego jest nazywany również atomem. Przykładowe tokeny są przedstawione w tabeli 1.

Tokeny, wzorce, leksemy

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Tabela: Tokeny, wzorce, leksemy

Token	Leksem	Wzorzec
if	if	if
id	abc, n, count,...	litera + cyfra
NUMBER	3.14, 1000	stała numeryczna
;	;	;

Tokeny, wzorce, leksemy

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Dla kodu:

```
printf ("Total = %d\n", score) ;
```

zarówno *printf* i *score* są leksemami pasującymi do wzorca tokenu *id*, natomiast *Total* = %*d**n* jest to leksem pasujący do literału (dosłowny tekst).

Tokeny, wzorce, leksemy

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

W wielu językach programowania następujące przypadki obejmują większość tokenów:

1. Jeden token dla każdego słowa kluczowego. Wzorzec dla słowa kluczowego jest taki sam jak słowo kluczowe.
2. Tokeny dla operatorów: indywidualny token dla każdego operatora lub jeden token dla grupy operatorów (przykładowo operatorów relacji).
3. Jeden token reprezentujący wszystkie identyfikatory.
4. Jeden lub większa liczba tokenów reprezentujących stałe, takie jak liczby i literały.
5. Tokeny dla każdego z symboli interpunkcyjnych, takich jak lewy i prawy nawias, przecinek, średnik.

Atrybuty tokenów

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Token posiada opcjonalne atrybuty. Najważniejszym przykładem jest token *id*, z którym musimy skojarzyć dużo informacji: typ danych, wymiar tablicy, liczbę elementów tablicy, miejsce w programie, w którym zmienna pojawia się po raz pierwszy.

Atrybuty tokenów. Atrybuty są przechowywane w tablicy symboli. Jednym z atrybutów jest zatem również wskaźnik do wpisu w tablicy symboli dla identyfikatora.

Atrybuty tokenów

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Przykład: Dla instrukcji przypisania w Fortranie:

```
E = M * C ** 2
```

mamy następujący wynik produkowany przez analizator leksykalny:

```
<id, pointer to symbol-table entry for E>
<assign_op>
<id, pointer to symbol-table entry for M>
<mult_op>
<id, pointer to symbol-table entry for C>
<exp-op>
<number, integer value 2>
```

Błędy leksykalne

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Błędy leksykalne (*lexical errors*):

- Wykrywanie błędów $fi(a == f(x))$...
- Raportowanie błędów.
- Usuwanie błędów.

Błędy leksykalne

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Usuwanie błędów. Załóżmy, że analizator leksykalny podczas rozpoznawania leksemu nie może kontynuować swojego działania, ponieważ żaden ze wzorców nie pasuje. Analizator może usunąć kolejne znaki z pozostałego wejścia, aż dopasuje wczytywane znaki do jakiegoś wzorca. Analizator leksykalny, który będziemy tworzyć, pozwala na rozpoznawanie liczb, identyfikatorów i „znaków białych” (spacji, tabulatorów i znaków nowej linii) w wyrażeniach.

Schemat translacji

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Schemat translacji:

```
expr ->expr + term          { print('+')}
      | expr - term         { print('-') }
      | term
term -> term * factor       { print('*') }
      | term / factor       { print('/')}
      | factor
factor -> ( expr )
        | num               { print (num.value) }
        | id                { print (id.lexeme) }
```

Schemat translacji dla białych znaków

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Większość języków umożliwia dowolną ilość przestrzeni białej między leksemami. Komentarze mogą być traktowane jako przestrzeń biała. Jeśli przestrzeń biała jest eliminowana przez analizator leksykalny, to parser nie będzie musiał brać pod uwagę znaków białych. Alternatywnie można uwzględnić znaki białe w gramatyce, ale to znacznie zwiększa złożoność parsera.

Schemat translacji dla białych znaków

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Pseudokod rozpoznawania i usuwania znaków białych:

```
for ( ; ; peek = next input character ) {
    if ( peek is a blank or a tab ) do nothing;
    else if ( peek is a newline ) line = line+1;
    else break;
}
```

Czytanie z wyprzedzeniem

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Czytanie z wyprzedzeniem. Analizator leksykalny może wymagać odczytywania znaków wejściowych z wyprzedzeniem, zanim zdecyduje, jaki ma być leksem właściwy. Na przykład analizator leksykalny dla C lub Javy po rozpoznaniu znaku $>$ musi odczytać następny znak. Jeśli następnym znakiem jest $=$, to znak $>$ jest częścią sekwencji znaków $>=$, reprezentujących leksem (operator) „większe lub równe”. Inaczej znak $>$ sam tworzy leksem „większy niż”; w takim przypadku analizator leksykalny odczytał jednak jeden znak za dużo. Ogólne podejście do czytania znaków wejścia z wyprzedzeniem jest oparte na zastosowaniu bufora wejściowego, z którego analizator leksykalny może odczytać znak i zapisać go z powrotem.

Czytanie z wyprzedzeniem

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Bufory wejściowe mogą być uzasadnione również w odniesieniu do efektywności analizatora, ponieważ pobieranie ciągu znaków jest zwykle bardziej wydajne niż odczyt jednego znaku na raz. Wskaźnik śledzi część wejścia, która już została przeanalizowana; przejście do poprzedniego znaku jest realizowane przez przesunięcie wskaźnika do tyłu. Jeden znak odczytany z wyprzedzeniem zazwyczaj wystarcza, więc prostym rozwiązaniem jest użycie zmiennej, powiedzmy o nazwie *peek*, do przechowywania następnego znaku wejściowego. Analizator leksykalny czyta z wyprzedzeniem tylko wtedy, gdy musi. Operatora $'*'$ można użyć, aby nie odczytywać następnego znaku. W takich przypadkach wartością zmiennej *peek* jest znak spacji, który może być pominięty, gdy analizator jest wywoływany, aby znaleźć następny leksem.

Czytanie z wyprzedzeniem

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Za każdym razem, gdy w wyrażeniu pojawia się pojedyncza cyfra, rozsądne wydaje się wczytywanie kolejnych cyfr w celu rozpoznania liczby całkowitej, ponieważ jest ona sekwencją cyfr. Stałe całkowite mogą być reprezentowane poprzez utworzenie symbolu terminalnego, powiedzmy o nazwie *num* dla każdej stałej, lub poprzez wprowadzenie składni stałych całkowitych do gramatyki. Praca łączenia cyfr w liczbę z reguły należy do zadań analizatora leksykalnego, w związku z czym liczby mogą być traktowane jako pojedyncze jednostki (leksemy) w trakcie parsowania i tłumaczenia kodu źródłowego.

Rozpoznawanie stałych

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Rozpoznawanie stałych. Gdy w strumieniu wejściowym pojawia się ciąg cyfr, analizator leksykalny przekazuje do parsera token, który składa się z terminala *num* wraz z atrybutem – rozpoznanej liczby całkowitej. Dla napisu wejściowego: $31 + 28 + 59$ analizator leksykalny produkuje następujący wynik:
$< num, 31 >< + >< num, 28 >< + >< num, 59 >$.

Rozpoznawanie stałych

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Pseudokod do rozpoznawania stałych zaprezentowano poniżej:

```
if ( peek holds a digit ) {
    v = 0;
    do {    v = v * 10 + integer value of digit peek;
            peek = next input character;
    } while ( peek holds a digit ) ;
    return token (num, v);
}
```

Rozpoznawanie identyfikatorów i słów kluczowych

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Rozpoznawanie identyfikatorów i słów kluczowych. Większość popularnych języków programowania używa słów kluczowych, na przykład: *for*, *do*, *if*, *while*, itd., które są reprezentowane przez ciągi znaków. Ciągi znaków są również wykorzystywane do tworzenia nazw zmiennych, tablic, funkcji itd. Chcąc uprościć parser, gramatyki traktują identyfikatory jako terminal, powiedzmy *id*, za każdym razem, gdy identyfikator pojawi się na wejściu.

Rozpoznawanie identyfikatorów i słów kluczowych

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Na przykład dla wejścia: $count = count + increment$; parser dostanie od analizatora leksykalnego następujący ciąg tokenów: $id = id + id$. Dla tokena id atrybutem jest leksem reprezentujący identyfikator.
Dla wejścia:
$count = count + increment$;
analizator leksykalny produkuje:
$< id, "count" >$
<=>
$< id, "count" >$
$< + >$
$< id, "increment" >< ; >$

Słowa kluczowe

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Słowa kluczowe generalnie spełniają zasady tworzenia identyfikatorów, a zatem jest potrzebny mechanizm do podjęcia decyzji: leksem reprezentuje słowo kluczowe czy identyfikator. Problem jest łatwiejszy do rozwiązania, jeśli słowa kluczowe są zastrzeżone – nie mogą one być wykorzystywane jako identyfikatory. Ciąg znaków tworzy identyfikator, jeżeli nie reprezentuje słowa kluczowego. Analizator leksykalny rozwiązuje ten problem za pomocą tablicy do przechowywania ciągów znaków, czyli symboli.

1. Tablica symboli może izolować resztę kompilatora od reprezentacji ciągów, fazy kompilatora mogą korzystać z referencji lub wskaźnika do łańcucha w tabeli. Referencje / wskaźniki mogą być bardziej efektywne niż manipulowanie samymi ciągami.
2. Tablica symboli może być inicjalizowana słowami zarezerwowanymi.

Słowa kluczowe

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Gdy analizator leksykalny rozpoznaje leksem, który może stanowić identyfikator, najpierw sprawdza, czy leksem jest przechowywany w tablicy symboli. Jeżeli jest przechowywany, to zwraca token przechowywany w tablicy; w przeciwnym razie tworzy i zwraca token, którego pierwszym elementem jest *id*. Tablica symboli może być zaimplementowana jako tablica haszująca z użyciem klasy o nazwie *Hashtable*. Na przykład: *Hashtable words = new Hashtable*();.

Słowa kluczowe

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

W poniższym fragmencie programu został przedstawiony podstawowy schemat działania analizatora leksykalnego i gromadzenia symboli w tablicy symboli:

```
if ( peek holds a letter ) {
        collect letters or digits into a buffer b;
        s = string formed from the characters in b;
        w = token returned by words.get(s);
        if ( w is not null )    return w;
        else {
                Enter the key-value pair  (s, <id, s
                    >) into words
                return token <id, s>;
        }
}
```

Ogólny schemat analizy leksykalnej

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Poniżej przedstawiony jest pseudokod funkcji *scan*, która zwraca tokeny:

```
Token scan() {
    skip white space;
    handle numbers;
    handle reserved words and identifiers;
    /* if we get here, treat read-ahead character
        peek as a token */
    Token t = new Token(peek);
    peek = blank
    return t;
```

Analizator leksykalny w C

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Analizator leksykalny w C:

```
int lineno = 1;
int tokenval = NONE;

int lexan ()
{
        int t;
        while (1) {
                t = getchar ();
                if (t == ' ' || t == '\t');
                else if (t == '\n')
                        lineno++;
```

Analizator leksykalny w C

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

```
                else if (isdigit (t))
                {
                        tokenval = t - '0';
                        t = getchar();
                        while (isdigit(t))
                        {
                                tokenval = tokenval
                                    *10 + t - '0';
                                t = getchar();
                        }
                        ungetc (t, stdin);
                        return NUM;
                }else
                {
                        tokenval = NONE;
                        return t;
                }
        }
}
```

Tablica symboli

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Tablica symboli jest wykorzystywana do przechowywania zmiennych oraz słów kluczowych. Jest inicjalizowana poprzez wstawienie do niej słów kluczowych:

```
int insert(const char* s, int t); /* zwraca indeks w
    tablicy symboli dla nowego leksemu s i tokenu t
    */
int lookup(const char* s); /* zwraca indeks wpisu
   dla leksemu s lub 0 gdy nie znaleziono */
```

```
insert("div", DIV);
insert("mod", MOD);
```

Tablica symboli

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Analizator leksykalny z tablicą symboli:

```
int lexan () {
        int t;
        while (1) {
                t = getchar ();
                if (t == ' ' || t == '\t');
                else if (t == '\n')
                        lineno++;
                else if (isdigit (t)) {
                        ungetc (t, stdin);
                        scanf ("%d", &tokenval);
                        return NUM;
```

Tablica symboli

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

```
        } else if (isalpha (t)) {
                int p, b = 0;
                while (isalnum (t)) {
                        lexbuf[b] = t;
                        t = getchar ();
                        b++;
                        if (b >= BSIZE)
                            error ("compiler
                            ␣error");
                }
                lexbuf[b] = EOS;
                if (t != EOF) ungetc (t,
                    stdin);
                p = lookup (lexbuf);
                if (p == 0) p = insert (
                    lexbuf, ID);
                tokenval = p;
                return symtable[p].token;
```

Tablica symboli

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

```
                    } else if (t == EOF) return DONE;
                    else {
                            tokenval = NONE;
                            return t;
                    }
            }
}
```

Ciągi znaków i języki

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Ciągi znaków i języki (*Strings and languages*). Alfabet jest to dowolny skończony zbiór symboli. Typowymi przykładami symboli są litery, cyfry i znaki interpunkcyjne. Zbiór $\{0, 1\}$ jest alfabetem binarnym. Znaki tablicy kodów ASCII tworzą ważny przykład alfabetu. Napis (ciąg znaków, łańcuch) nad pewnym alfabetem jest to skończony ciąg symboli z tego alfabetu. Długość napisu s, zapisywana jako $|s|$, jest to liczba wystąpień dowolnych symboli w napisie s. Na przykład napis *banan* ma długość 5. Pusty ciąg znaków, oznaczany jako ε, jest ciągiem o zerowej długości.

Ciągi znaków i języki

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Bardzo szeroka definicja języka: Język jest to zbiór napisów nad pewnym ustalonym alfabetem.
Przykład: Dla alfabetu $L = A, ..., Z$, zbiór $A, B, C, BF, \ldots, ABZ, \ldots$ jest językiem zdefiniowanym przez alfabet L. Należy zauważyć, że definicja „języka” nie wymaga, aby każdy napis miał jakiś sens.

Ciągi znaków i języki

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Pojęcia związane z napisami

Prefiks (przedrostek) napisu s jest to dowolny napis uzyskany przez usunięcie zera lub większej liczby symboli z końca s. Na przykład *ban*, *banana* i ε są prefiksami napisu *banana*.

Sufiks (przyrostek) napisu s jest to dowolny napis uzyskany przez usunięcie zera lub większej liczby symboli z początku s. Na przykład *nana*, *banana* i ε są sufiksami napisu *banana*.

Podnapis (podciąg spójny) napisu s jest uzyskiwany poprzez usuwanie jakichkolwiek przedrostków i przyrostków z s. Na przykład *banana*, *nan* i ε są podnapisami napisu *banana*.

Ciągi znaków i języki

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Złączenie (*concatenation*) napisów x i y jest to napis xy utworzony przez dodanie y na końcu do x. Na przykład jeśli $x = dog$ i $y = house$, to $xy = doghouse$. Dla napisu pustego są prawidłowe równości: $\varepsilon s = s\varepsilon = s$.

Podnoszenie napisu do potęgi definiujemy w następujący sposób: $s^0 = \varepsilon \quad s^i = s^{i-1}s$ dla $i > 0$.

Przykłady: $s^1 = s, \quad s^2 = ss, \quad s^3 = sss$.

Ciągi znaków i języki

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Operacje na językach:

Suma(*union*):

$L \cup M = s | s \in L \text{ or } s \in M$

Złączenie (*concatenation*):

$LM = xy | x \in L \text{ and } y \in M$

Potęgowanie (*exponentiation*):

$L^0 = \varepsilon; L^i = L^{i-1}L$

Domknięcie Kleene'a (*Kleene closure*):

$L^* = \cup_{i=0,\ldots,\infty} L^i$

Domknięcie dodatnie (*positive closure*):

$L^+ = \cup_{i=1,\ldots,\infty} L^i$

Wyrażenia regularne

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Wyrażenia regularne (*regular expressions*). Podstawowe wyrażenia regularne:

- ε jest wyrażeniem regularnym oznaczającym język $\{\varepsilon\}$
- $a \in \Sigma$ jest wyrażeniem regularnym oznaczającym język $\{a\}$

Wyrażenia regularne

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Jeśli r i s są wyrażeniami regularnymi oznaczającymi języki $L(r)$ i $M(s)$, to:

- $r|s$ jest wyrażeniem regularnym oznaczającym język $L(r) \cup M(s)$,
- rs jest wyrażeniem regularnym oznaczającym język $L(r)M(s)$,
- r^* jest wyrażeniem regularnym oznaczającym język $L(r)^*$,
- (r) jest wyrażeniem regularnym oznaczającym język $L(r)$.

Wyrażenia regularne

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Czyli wyrażenia regularne są budowane z mniejszych wyrażeń regularnych w sposób rekurencyjny. Wyrażenia regularne mogą zawierać zbędne pary nawiasów. Możemy usunąć niektóre pary nawiasów, jeśli przyjmiemy następującą konwencję:

- Operator jednoargumentowy * (operator domknięcia) ma najwyższy priorytet i jest łączny lewostronnie.
- Złączenie ma drugi najwyższy priorytet i jest łączne lewostronnie.

Wyrażenia regularne

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Przykład: $\Sigma = a, b$.

1. Wyrażenie regularne $a|b$ oznacza język $\{a, b\}$.
2. $(a|b)(a|b)$ oznacza aa, ab, ba, bb, czyli język, którego elementami są napisy o długości 2.
3. a^* oznacza język, którego elementy są złożeniem zera lub większej liczby symboli a: $\{\varepsilon, a, aa, aaa, ...\}$.
4. $(a|b)^*$ oznacza język, którego elementy są złożeniem zera lub większej liczby symboli a lub b: $\{\varepsilon, a, b, aa, ab, ba, bb, aaa, ...\}$.
5. $a|a * b$ oznacza język $\{a, b, ab, aab, aaab, ...\}$

Wyrażenia regularne

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Przykłady:

Rozważmy alfabet $\Sigma = \{a\}$

Jakie jest wyrażenie regularne, które oznacza język, którego każde słowo ma długość nieparzystą?

Rozważmy alfabet $\Sigma = \{a\}$

$a(aa)*$ jest wyrażeniem regularnym, które oznacza język, którego każde słowo ma długość nieparzystą.

Rozważmy alfabet $\Sigma = \{a, b\}$

Jakie jest wyrażenie regularne, które oznacza język, którego każde słowo zaczyna się od litery b?

Rozważmy alfabet $\Sigma = \{a, b\}$

$b(a|b)^*$ jest wyrażeniem regularnym, które oznacza język, którego każde słowo zaczyna się od litery b.

Wyrażenia regularne

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Rozważmy alfabet $\Sigma = \{a, b\}$
Jakie jest wyrażenie regularne, które oznacza język, którego każde słowo musi zaczynać się na literę a, a kończy się na literę b?
Rozważmy alfabet $\Sigma = \{a, b\}$
$b(a|b)^*a$ jest wyrażeniem regularnym, które oznacza język, którego każde słowo musi zaczynać się na literę b, a kończy się na literę a.
Rozważmy alfabet $\Sigma = \{a, b, c\}$
Jakie jest wyrażenie regularne, które oznacza język:
$L = \{a, c, ab, cb, abb, cbb, abbb, cbbb, abbbb, cbbbb, ...\}$?
Rozważmy alfabet $\Sigma = \{a, b, c\}$
$((a|c)b^*)$ oznacza język:
$L = \{a, c, ab, cb, abb, cbb, abbb, cbbb, abbbb, cbbbb, ...\}$.

Definicje regularne

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Definicje regularne (*regular definitions*). Jeśli Σ jest alfabetem symboli podstawowych, to definicją regularną jest sekwencja :

$$d_1 -> r_1$$

$$d_2 -> r_2$$

...

$$d_n -> r_n$$

gdzie:

1. Każde d_j jest to unikatowy symbol (d_j są różnymi symbolami), nienależący do alfabetu.
2. Każde r_i jest wyrażeniem regularnym nad symbolami z alfabetu $\{d_1, d_2, ..., d_i - 1\}$.

Definicje regularne

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Definicje regularne nie mogą być rekurencyjne:

```
digits -> digit digits | digit
```

Powyższe to błąd!

Definicje regularne

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

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.

Rozszerzenia wyrażeń regularnych

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Rozszerzenia wyrażeń regularnych. Rozszerzenia, które zostały po raz pierwszy wprowadzone do narzędzia Lex:

1. Jedno wystąpienie lub większa liczba wystąpień. Jednoskładnikowy postfiksowy operator $+$ reprezentuje domknięcie dodatnie wyrażenia regularnego i jego języka. Jeśli r jest wyrażeniem regularnym, to $(r)+$ oznacza język $(L(r))+$. Operator $+$ (domknięcie dodatnie) ma ten sam priorytet i łączność co operator $*$ (domknięcie) oznaczający zero lub większą liczbę znaków. Dwa użyteczne prawa algebraiczne:

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

Rozszerzenia wyrażeń regularnych

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Rozszerzenia wyrażeń regularnych. Rozszerzenia, które zostały po raz pierwszy wprowadzone do narzędzia Lex:

2. Zero wystąpień lub jedno wystąpienie. Jednoskładnikowy postfiksowy operator ? oznacza „zero lub jedno wystąpienie". To oznacza, że $r?$ jest równoważne $r|\varepsilon$ lub innymi słowy: $L(r?) = L(r) \cup \varepsilon$. Operator ? ma ten sam priorytet i łączność co operatory $*$ i $+$.
3. Wyrażenie regularne $a_1|a_2|...|a_n$, gdzie a_i są symbolami alfabetu, może być zastąpione przez skrót $[a_1 a_2 ...]$. Co więcej, gdy $a_1, a_2, ..., a_n$ tworzą logiczny ciąg, np. kolejne litery duże, litery małe lub cyfry, możemy zastąpić je przez $a_1 - a_n$: $[a - z] = ab...z$

Rozszerzenia wyrażeń regularnych

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Przykład: Definicje regularne jak niżej:

```
letter -> A | B |...| Z | a | b |...| z |...
digit -> 0| 1 |...| 9
id -> letter_ (letter_ | digit )*
```

są podstawową formą zapisu.

Rozszerzenia wyrażeń regularnych

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Definicje regularne możemy zapisać również w skróconej postaci:

```
letter_  -> [A- Za -z_]
digit -> [0-9]
id -> letter_( letter_ | digit )*
```

Wyrażenia regularne

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Rozważmy następujący przykład:

```
stmt -> if expr then stmt
      |  if expr then stmt else stmt
      |  .

expr -> term relop term
      |  term

term -> id
      | number
```

Wzorce leksemów

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Wzorce leksemów są przedstawione poniżej:

```
digit -> [0-9]
digits -> digit+
number ->digits (.digits)? ( E [+-]? digits )?
Letter-> [A-Za-z]
id  -> letter ( letter I digit )*
if -> if
then -> then
else -> else
relop -> < | > | <= | >= | = | <>
```

Wzorce leksemów

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

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.

Wzorce leksemów

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

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. W tabeli przedstawione zostały leksemy i tokeny oraz odpowiadające im wartości atrybutów.

Wzorce leksemów

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Tabela: 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, ...

Diagramy przejść

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

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. Każdy stan reprezentuje warunek, który może wystąpić podczas procesu skanowania strumienia wejściowego w celu rozpoznawania leksemu, który pasuje do jednego z kilku wzorców. 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 symbolem a.

Diagramy przejść

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Jeśli znajdziemy taką krawędź, to przechodzimy do stanu, do którego prowadzi ta krawędź. Zakładamy, że wszystkie nasze diagramy przejść są deterministyczne, co oznacza, że taka sama etykieta nie może oznaczać dwóch lub większej liczby krawędzi wychodzących z tego samego stanu.

Diagramy przejść

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Na rysunku pokazany jest diagram przejść dla tokenu relop.
Ważne pojęcia związane z diagramami przejść:

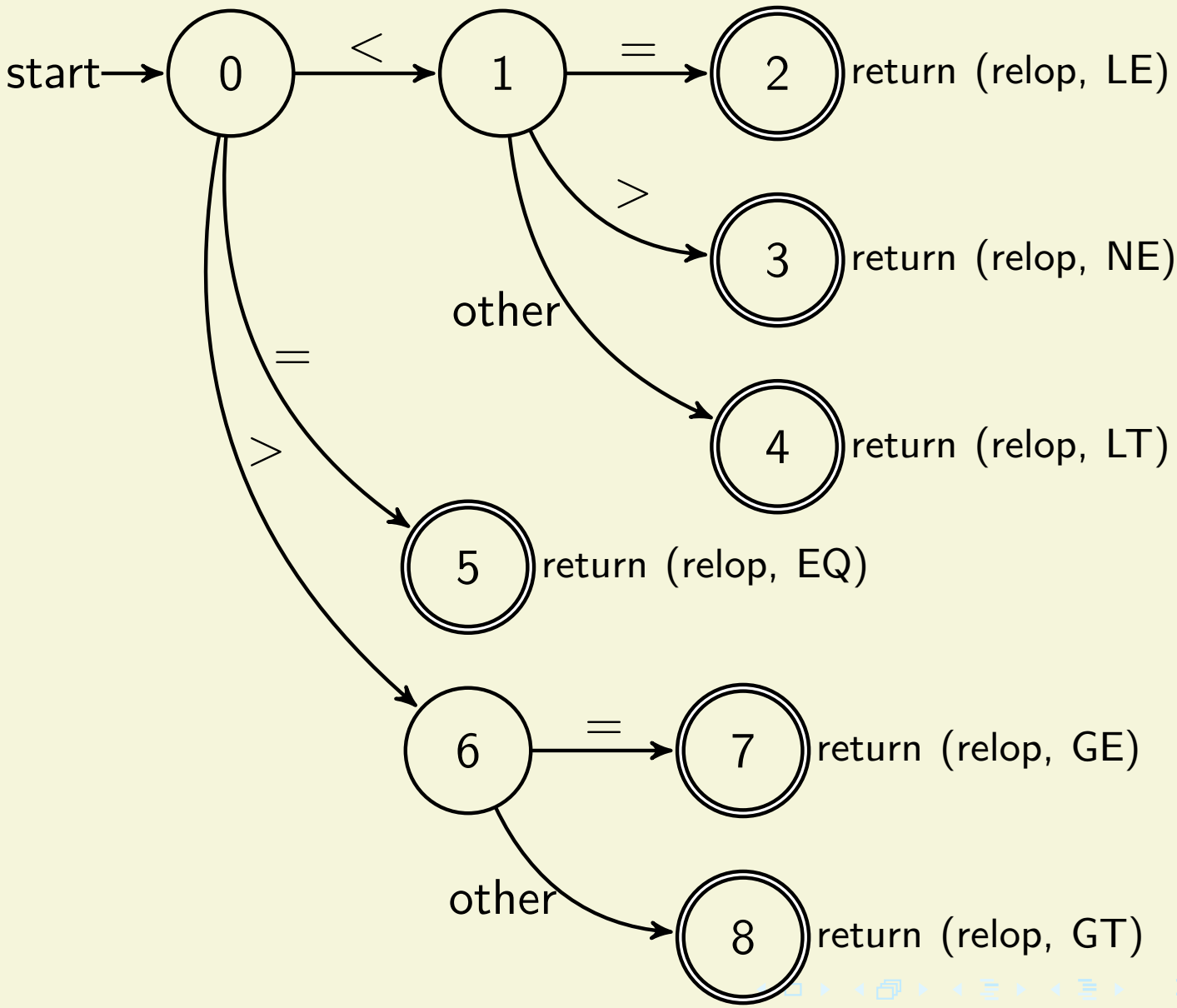
1. Stany końcowe, które wskazują, że leksem został rozpoznany; są one oznaczane na rysunkach okręgiem podwójnym. Jeśli ma być wykonana jakaś akcja (zazwyczaj zwrot tokena do parsera), to dołączamy tę akcję do stanu akceptującego.
2. Ponadto, jeśli jest to konieczne, aby przesunąć wskaźnik symbolu o jedną pozycję do przodu (co oznacza, że leksem nie zawiera symbolu, który prowadzi do stanu akceptującego), to należy dodatkowo wstawić znak $*$ obok stanu końcowego.
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.

Diagramy przejść

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

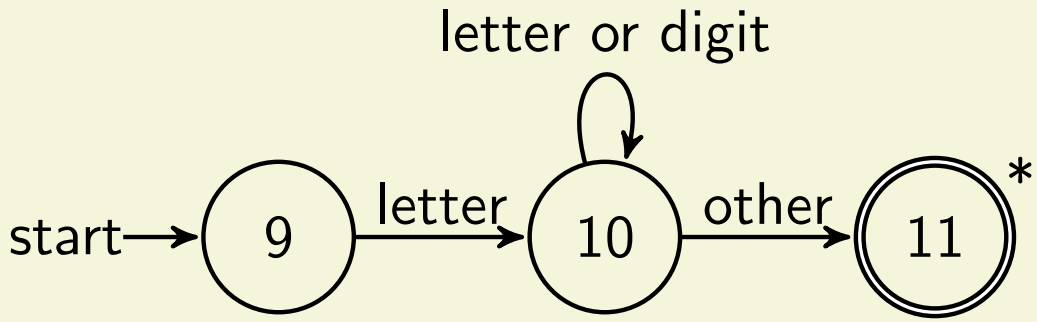


Diagramy przejść

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Rysunek pokazuje schemat rozpoznawania nazwy, natomiast następny rysunek – schemat rozpoznawania znaków białych.



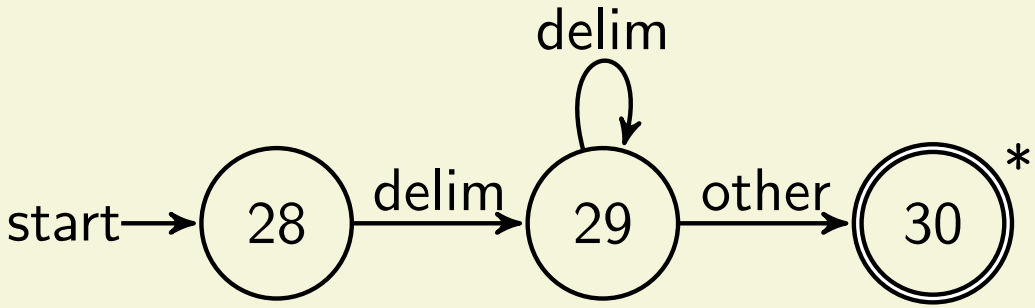
Rysunek: Rozpoznawanie nazwy

Diagramy przejść

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA



Rysunek: Rozpoznawanie znaków białych

Implementacja analizatora leksykalnego

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Implementacja analizatora leksykalnego. Możemy wprowadzić zmienną o nazwie *state* do przechowywania bieżącego stanu diagramu przejść. Wtedy możemy zaimplementować analizator leksykalny na podstawie konstrukcji *switch*.

Implementacja analizatora leksykalnego

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Implementacja diagramu przejść. Każdy stan zawiera odpowiedni segment kodu. Jeśli są jakieś 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.

Implementacja analizatora leksykalnego

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

```
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;
        }
}
```

Implementacja analizatora leksykalnego

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Liczby określające poszczególne stany (zmienna *state*) w powyższym kodzie źródłowym odpowiadają stanom z rysunków.

(f)Lex

Metody kompilacji

Analiza leksykalna

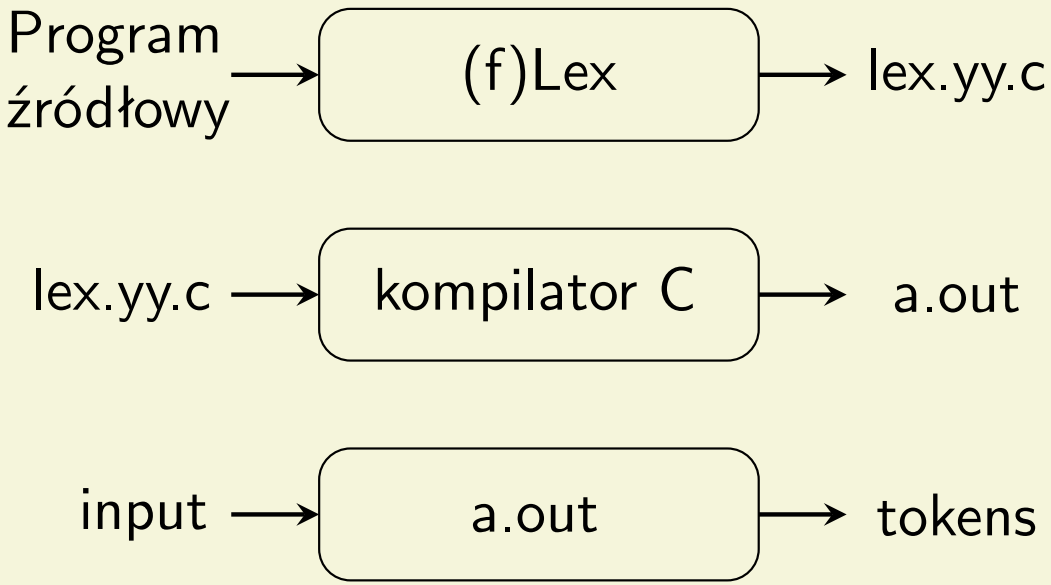
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Rysunek pokazuje sposób użycia narzędzia do analizy leksykalnej Lex. Program wejściowy dla narzędzia Lex tworzymy w języku Lex, natomiast samo narzędzie nazywamy kompilatorem Lex. Kompilator Lex konwertuje tokeny wejściowe na diagram przejść oraz generuje kod symulujący diagram przejść, który domyślnie jest zapisywany w pliku o nazwie *lex.yy.c*.

(f)Lex

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA



Rysunek: Narzędzie do analizy leksykalnej Lex

(f)Lex

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Struktura programu w języku Lex jest następująca:

```
deklaracje
%%
reguły translacji
%%
funkcje pomocnicze
```

(f)Lex

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Sekcja deklaracji zawiera deklaracje zmiennych, identyfikatory zadeklarowane jako stałe, np. nazwę tokena oraz definicje regularne. Reguły translacji mają postać: *wzorzec* {*akcja*} *(pattern* {*action*}*)*. Każdy wzorzec jest wyrażeniem regularnym, które może korzystać z definicji regularnych określonych w sekcji deklaracji. Akcje są fragmentami kodu, zwykle napisanego w języku C, choć modyfikacje narzędzia Lex używają innych języków. Trzecia sekcja zawiera dodatkowe funkcje, które są stosowane w akcjach. Alternatywnie funkcje te mogą być kompilowane osobno i ładowane za pomocą analizatora leksykalnego.

(f)Lex

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

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

(f)Lex

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Rozwiązywanie konfliktów w Lex. Lex stosuje dwie zasady w celu wybrania właściwego leksemu, gdy kilka prefiksów pasuje do jednego lub większej liczby wzorców:

1. Wybiera najdłuższy prefiks.
2. Jeśli najdłuższy prefiks pasuje do dwóch lub więcej wzorców, wybiera wzorzec wcześniejszy w tekście programu Lex.

(f)Lex

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Przykład: Pierwsza zasada mówi nam, żeby czytać dalej litery i cyfry, chcąc znaleźć najdłuższy prefiks grupy znaków pasujących do identyfikatora. Mówi nam także, żeby potraktować ciąg znaków "¡=" jako jeden leksem. Druga zasada sprawia, że rozpoznane słowo jest traktowane jako słowo kluczowe, jeżeli deklarujemy listę słów kluczowych przed deklaracją identyfikatora w programie Lex. Na przykład, jeśli słowo *then* ma najdłuższy prefiks, który pasuje do wzorca, oraz jeżeli *then* występuje wcześniej niż $\{id\}$, to jest zwracany token THEN (nie ID) .

(f)Lex

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Przykładowy plik dla programu Lex:

```
%{
#include <stdio.h>
#include "y.tab.h"
%}

%%
\=\=            {return EQ;}
\!\=            {return NEQ;}
\<\=            {return LEQ;}
\>\=            {return GEQ;}
\<                      {return '>';}
\>                      {return '<';}
\(                      {return '(';}
\)                      {return ')';}
\!                      {return '!';}
\=                      {return '=';}
\+                      {return '+';}
```

(f)Lex

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Przykładowy plik dla programu Lex:

```
\-                          {return '-';}
\*                          {return '*';}
\/                          {return '/';}
\;                          {return ';';}
"int"               {return INT;}
"if"                {return IF;}
"else"              {return ELSE;}
[A-Za-z_][A-Za-z0-9_]*   {return ID;}
[1-9][0-9]*|0       {return NUM;}
.                           {yyerror("Error");
%%
```

(f)Lex

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

W powyższym przykładzie następujące linie mają odpowiednie znaczenie. Linia 1 rozpoczyna sekcję nagłówkową, z której kod jest kopiowany bezpośrednio do kodu wyjściowego. Następnie linia 2 dołącza plik nagłówkowy z definicjami funkcji języka C. Linia 3 dołącza plik nagłówkowy zawierający definicje tokenów. Definicje te mogą być wygenerowane przez inny program, np. YACC lub bison, ewentualnie w przypadku prostego analizatora mogą być one zdefiniowane ręcznie. Linia 4 to zakończenie sekcji nagłówkowej. Linia 5 rozpoczyna sekcję z regułami translacji; w następnych liniach są zdefiniowane reguły dopasowania leksemów.

(f)Lex

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Linie od 6 do 11 definiują reguły dla operatorów porównania, natomiast linie 12–17 – leksemy dla operatorów służących do budowania wyrażeń arytmetycznych. Linie 18–20 to przykładowe definicje słów kluczowych języka. Linia 21 definiuje leksem identyfikatora, a linia 22 opisuje leksemy liczb całkowitych. Linia 23 zgłasza błąd w przypadku braku dopasowania znaków na wejściu do wcześniejszych definicji leksemów. Linia 24 kończy tę sekcję; po tej linii może wystąpić kod w języku C, w tym kod funkcji *main* oraz funkcji pomocniczych (np. *yyerror*). Kod ten zostanie, podobnie jak sekcja nagłówkowa, skopiowany do kodu wyjściowego.

(f)Lex

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Leksemy zdefiniowane w powyzszym programie pozwalają na przetworzenie prostego języka podobnego do języka C umożliwiającego definiowanie instrukcji warunkowych, prostych wyrażeń i operowania tylko na liczbach całkowitych.

Automaty skończone

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Automaty skończone. Zajmiemy się teraz pytaniem: Co robi Lex z programem wejściowym? Opierając się na programie wejściowym, Lex tworzy automat skończony (*finite automata*).

Automaty skończone

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Automat skończony jest podobny do diagramu przejść z kilkoma różnicami:

1. Automaty skończone służą tylko do rozpoznawania, po prostu zwracają odpowiedź „tak (rozpoznany)” lub „nie (nierozpoznany)” w stosunku do ciągu wejściowego.
2. Automaty skończone dzielimy na:
   1. niedeterministyczne (*nondeterministic finite automata* – NFA), które nie mają ograniczeń na etykiety krawędzi; tym samym symbolem można oznaczyć kilka krawędzi wychodzących z tego samego stanu; napis pusty może być etykietą krawędzi.
   2. deterministyczne (*deterministic finite automata* – DFA), w których każda krawędź, wychodząca z jakiegoś stanu, ma unikatową etykietę.

Automaty skończone

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Zarówno deterministyczne automaty skończone, jak i automaty niedeterministyczne są w stanie rozpoznać te same języki oparte na wyrażeniach regularnych.

Automaty skończone

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Niedeterministyczny automat skończony zawiera:

1. Skończony zbiór stanów S.
2. Alfabet wejściowy Σ. Zakładamy, że napis pusty ε nie należy do alfabetu Σ.
3. Funkcję przejścia, która określa dla każdego stanu i dla każdego symbolu należącego do zbioru $\Sigma \cup \{\varepsilon\}$ zbiór kolejnych stanów, do których można przejść.
4. Jeden stan początkowy (startowy) $s0$.
5. Zbiór stanów końcowych (akceptowalnych) F, który jest podzbiorem zbioru S.

Automaty skończone

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Możemy reprezentować zarówno NFA, jak i DFA przez graf przejść, w którym węzły reprezentują stany, natomiast krawędzie oznakowane reprezentują funkcję przejścia. 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. Graf ten jest bardzo podobny do diagramu przejść, z następującymi wyjątkami:

- ten sam symbol może oznaczyć kilka krawędzi wychodzących z jednego stanu do kilku różnych stanów;
- krawędź może być oznakowana za pomocą napisu pustego, który oznaczamy jako ε.

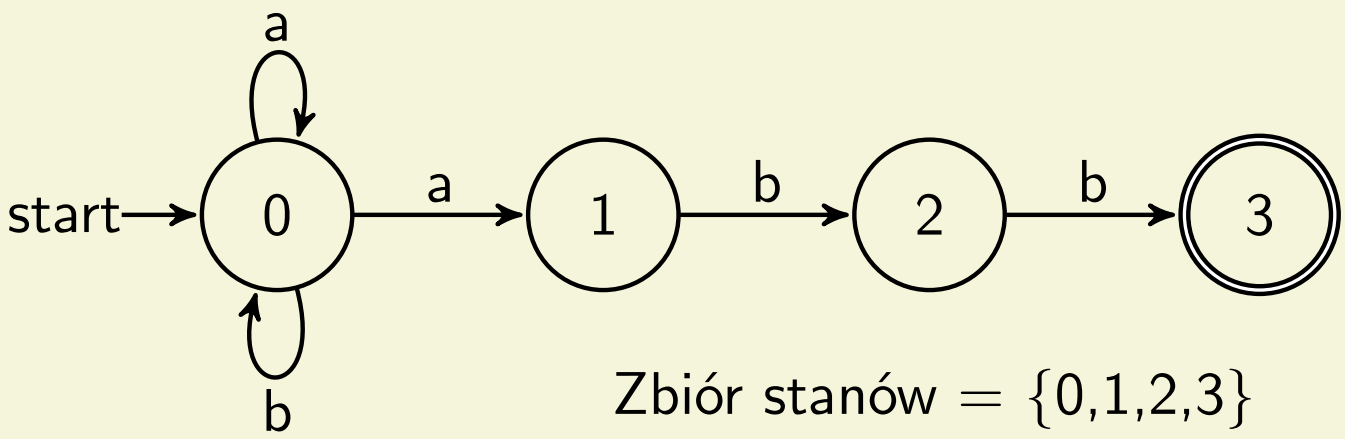
Automaty skończone

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Przykładowy niedeterministyczny automat skończony pokazuje rysunek .



Zbiór stanów = {0,1,2,3}

Język = {a,b}

Stan startowy S0

Stan końcowy S3

Rysunek: Przykład NFA: rozpoznawanie wyrażenia regularnego $(a|b) * abb$

Tablice przejść

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Tablice przejść. Możemy również reprezentować NFA przez tablicę przejść, której wiersze odpowiadają stanom, natomiast kolumny odpowiadają symbolom wejściowym Σ i ε. Wpis dla danego stanu i wejścia jest to wartość zwracana przez funkcję przejścia dla tych argumentów. Jeśli jakiś element tablicy zawiera symbol $\varnothing$, oznacza to, że taki element nie zawiera żadnej informacji. Przykładowa funkcja przejścia została przedstawiona w tabeli.

Tablice przejść

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Tabela: **Funkcja przejścia** przedstawiona w formie tabeli

	Wejście	
Stan	a	b
0	0,1	0
1	$\varnothing$	2
2	$\varnothing$	3

NFA akceptuje ciąg znaków x wtedy i tylko wtedy, gdy istnieje jakaś ścieżka w grafie przejść od stanu początkowego do jednego ze stanów końcowych – takiego, że symbole wzdłuż ścieżki tworzą x. Należy pamiętać, że etykiety oznakowane jako ε są ignorowane.

Tablice przejść

Metody kompilacji

Analiza leksykalna
Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Deterministyczne automaty skończone. Jeśli używamy tablicy przejścia do reprezentowania DFA, to każdy pojedynczy wpis reprezentuje pojedynczy stan; dlatego zapisujemy ten stan bez nawiasów. DFA pozwala na bardzo łatwe rozpoznawanie leksemów. Każde wyrażenie regularne i każdy NFA mogą być konwertowane do DFA akceptującego ten sam język. Na rysunku 7 pokazany jest diagram stanów dla $(a|b)*abb$.

Algorytm: symulacja DFA

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Algorytm: symulacja DFA

Wejście: Napis x, który kończy się znakiem *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

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

Symulacja DFA:

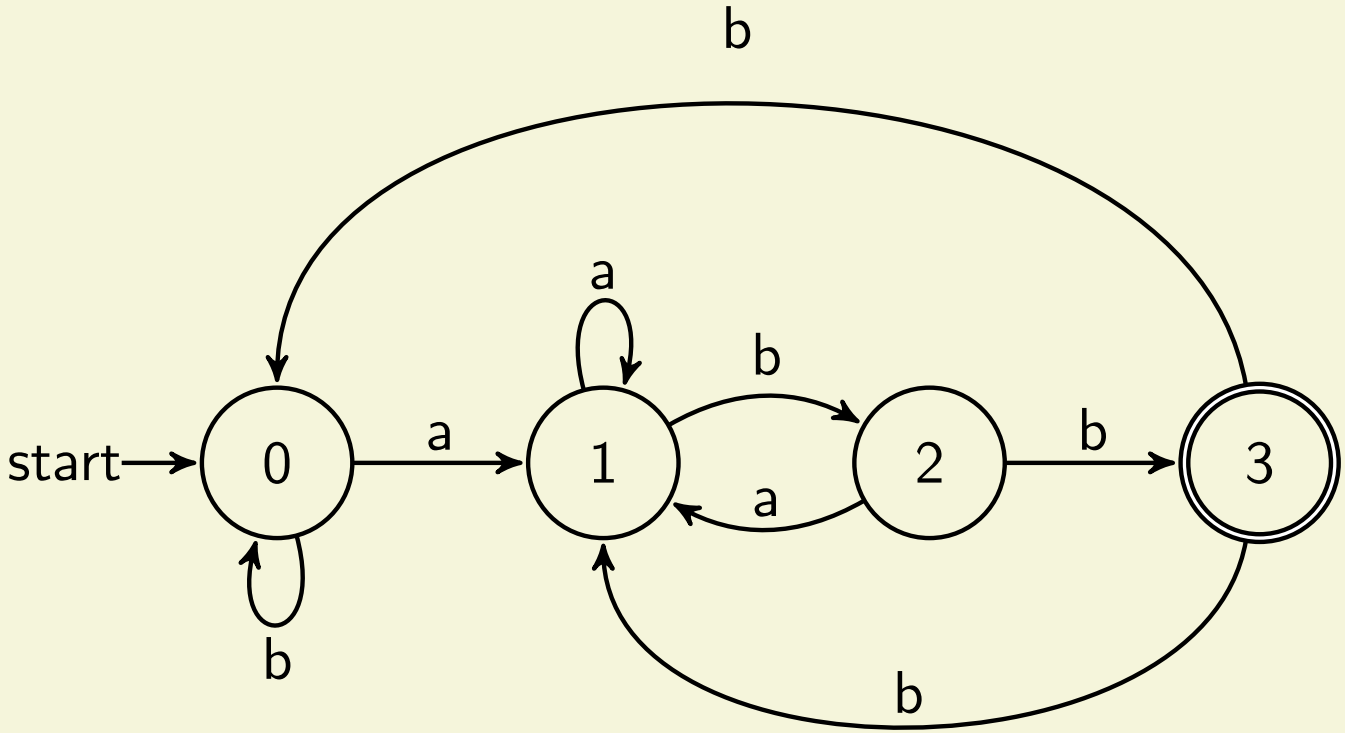
```
begin
        s := s0;
        c := nextchar;
        while c <> eof do
        begin
                s := move(s, c); // funkcja przejść
                c := nextchar
        end;
        if s is in F then
                return "yes"
        else
                return "no"
        end;
end.
```

Przykład

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA



Rysunek: Przykład $(a|b) * abb$

Konwersja RE na NFA

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

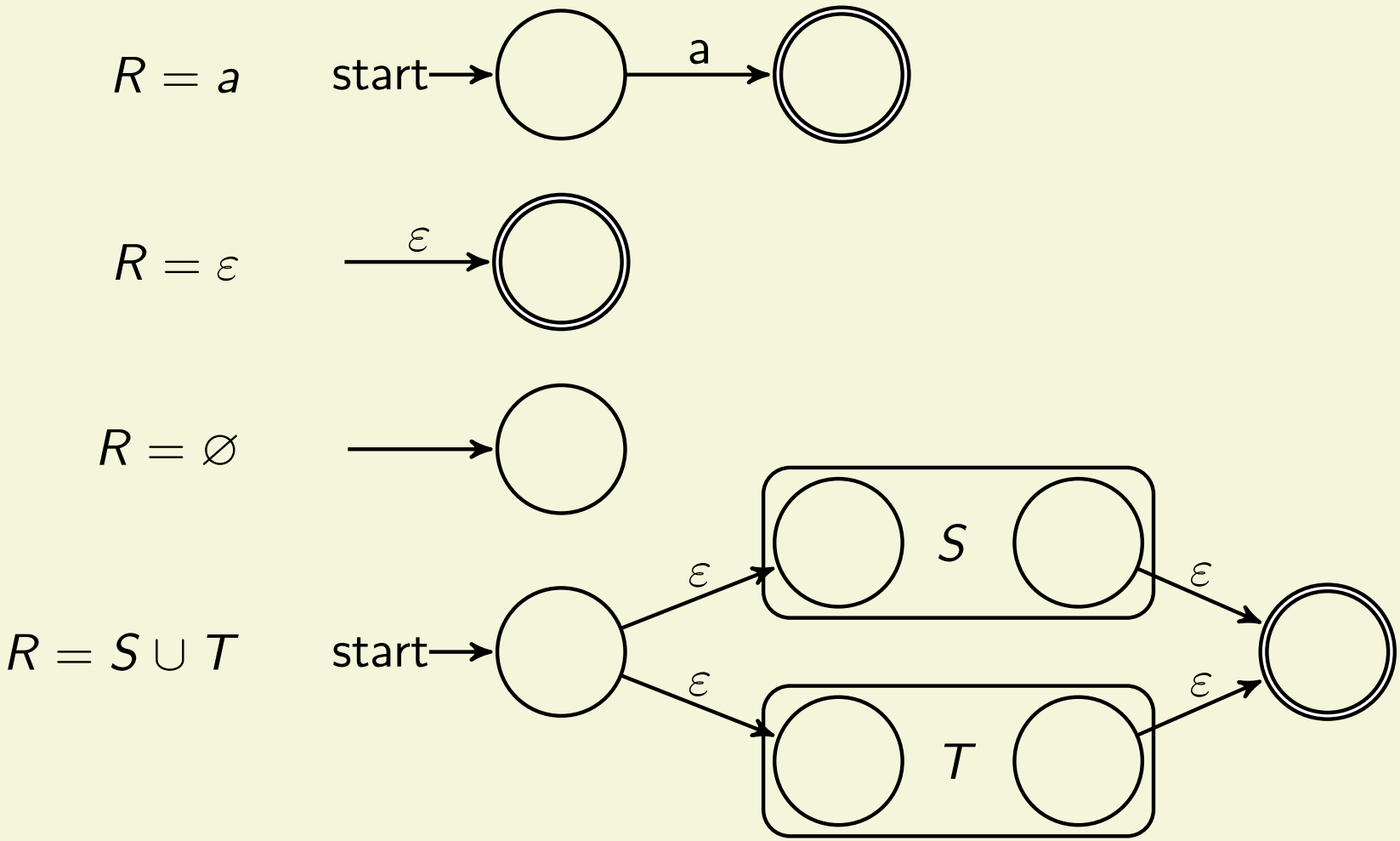
Konwersja RE na NFA. RE zawsze może być przekonwertowane na NFA . W tym celu korzysta się z konwersji bazowych pokazanych na pierwszym rysunku. Natomiast na kolejnym rysunku przedstawiono zamianę $R = (ab + a)*$ na NFA. Konwersja rozpoczyna się od najprostszych elementów.

Konwersja RE na NFA

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA



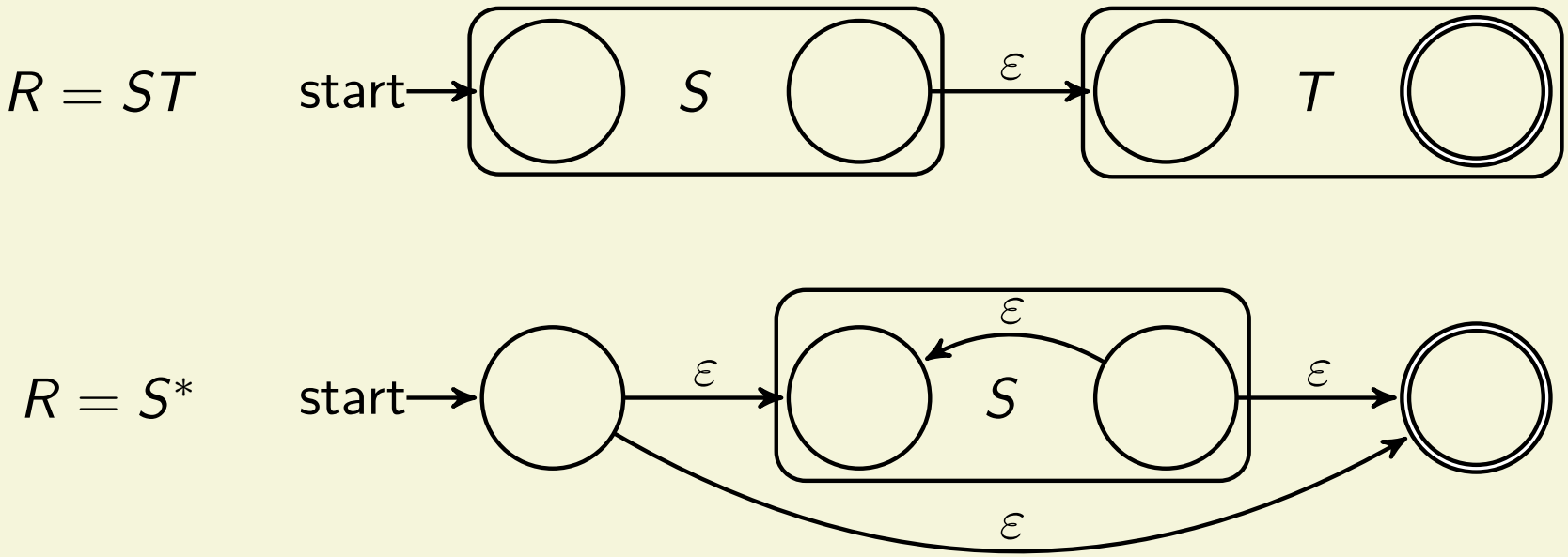
Rysunek: Konwersja RE na NFA – konwersje bazowe

Konwersja RE na NFA

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA



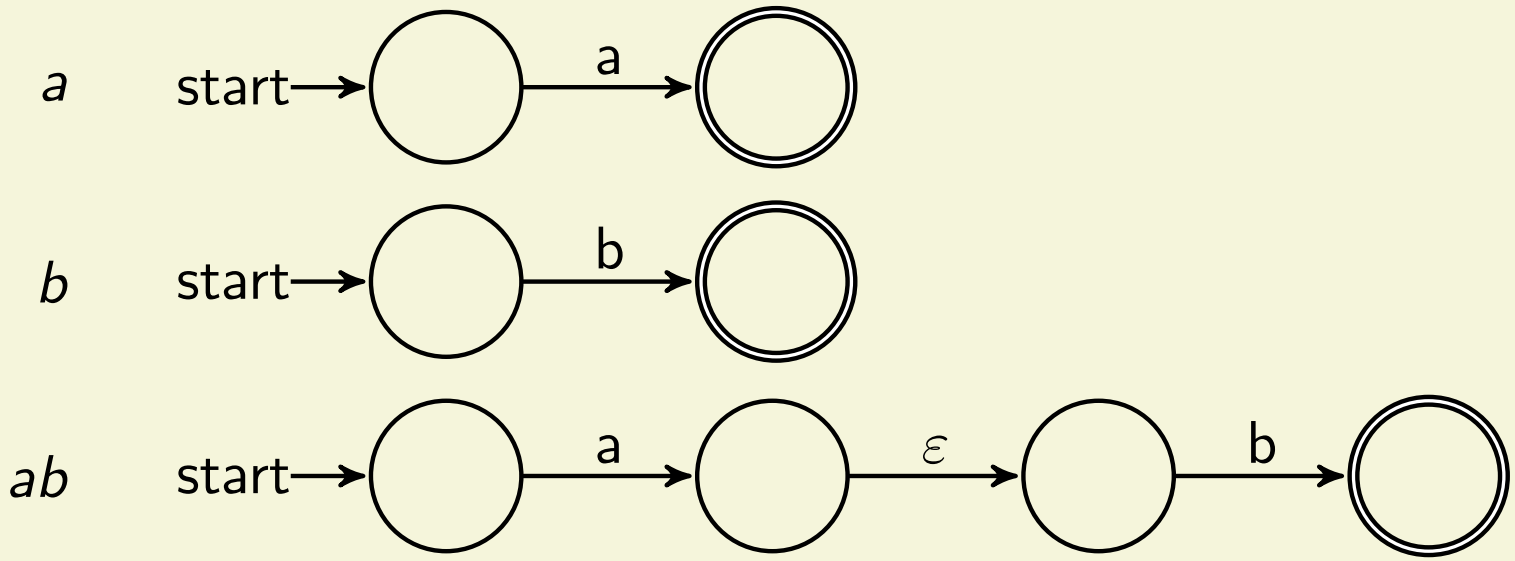
Rysunek: Konwersja RE na NFA – konwersje bazowe

Konwersja RE na NFA

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA



Rysunek: Konwersja RE na NFA – przykład

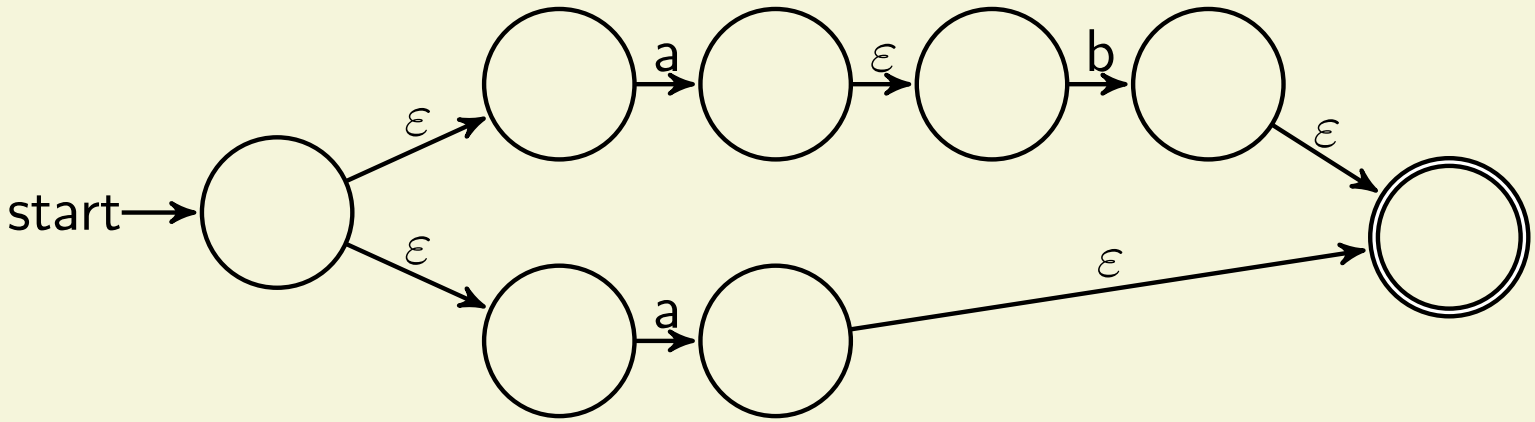
Konwersja RE na NFA

Metody kompilacji

Analiza leksykalna

Tokeny, wzorce, leksemy
Atrybuty tokenów
Błędy leksykalne
Schemat translacji
Czytanie z wyprzedzeniem
Rozpoznawanie stałych.
Rozpoznawanie identyfikatorów
Rozpoznawanie słów kluczowych
Ciągi znaków i języki
Wyrażenia regularne
Definicje regularne
Rozszerzenia
Wzorce leksemów
Diagramy przejść
Implementacja analizatora leksykalnego
Automaty skończone
Tablice przejść
Symulacja DFA
Konwersja RE na NFA

$ab + a$



$(ab + a)^*$