2 1976

prace

Instytutu Maszyn Matematycznych

rok XVIII

Rok XVIII 2 1976

prace

Instytutu

Maszyn

Matematycznych

Roman KRZEMIEŃ

KONSTRUKCJA ANALIZATORA LEKSYKALNEGO W METATRANSLATORZE

Praca doktorska napisana
pod kierunkiem prof. dr hab.W.M.Turskiego

Zjednoczenie Przemysłu Automatyki i Aparatury Pomiarowej "MERA"
INSTYTUT MASZYN MATEMATYCZNYCH

Adres Redakcji: Instytut Maszyn Matematycznych
Branżowy Ośrodek INTE
ul. Krzywickiego 34, 02-078 WARSZAWA
tel. 28-37-29

Druk IMM z. 43/76 n. 450 egz. pap. piśmienny kl. III g. 70. GP-II-1435/76

Przedstawiona Czytelnikom praca doktorska
ROMANA KRZEMIENIA została wyróżniona
nagrodą Ministra Przemysłu Maszynowego
w 1976 r.

Spis treści

Prace IMM
Zeszyt 2

519.688.3:519.682:801.56

Leksykograf zajmuje się korelowaniem form językowych w pewien szczególny sposób, mianowicie łączeniem w pary synonimów.

... leksykograf chce wiedzieć jakie formy są synonimiczne, czyli tożsame pod względem znaczenia.

("Z punktu widzenia logiki". Quine W.V.)

KONSTRUKCJA ANALIZATORA LEKSYKALNEGO W METATRANSLATORZE

Roman KRZEMIEŃ

Pracę złożono 20.09.1975

W pracy przedstawiono nową metodę konstrukcji analizatorów leksykalnych w metatranslatorach. Proponowana metoda jest całkowicie różna od metod stosowanych dotychczas, ponieważ atomy leksykalne są znajdowane automatycznie jedynie na podstawie opisu składu języka.

1. WSTĘP

Analizę leksykalną można traktować jako wstępną analizę syntaktyczną. Program dokonujący analizy leksykalnej (L-analizy) czyta tekst w języku źródłowym, rozumiany jako ciąg znaków, i grupom znaków przyporządkowuje pewne elementarne jednostki syntaktyczne, zwane atomami leksykalnymi.

Program ten, zwany analizatorem leksykalnym lub L-analizatorem korzysta jednak ze znacznie prostszego i efektywniej-

szego aparatu lingwistycznego niż program dokonujący analizy syntaktycznej, zwany S-analizatorem lub analizatorem syntaktycznym.

Składnię atomów leksykalnych można bowiem zapisać za pomocą gramatyki regularnej (np. <identyfikator> ::= litera | <identyfikator> litera | <identyfikator> cyfra, gdzie przyjmujemy, że litera i cyfra są symbolami terminalnymi). Do rozbioru atomów leksykalnych można zatem użyć automatu skończenie stanowego (zamiast automatu ze stosem), co znacznie przyspiesza analizę. Łatwość zaprogramowania automatu skończonego umożliwia również automatyczną konstrukcję L-analizatora.

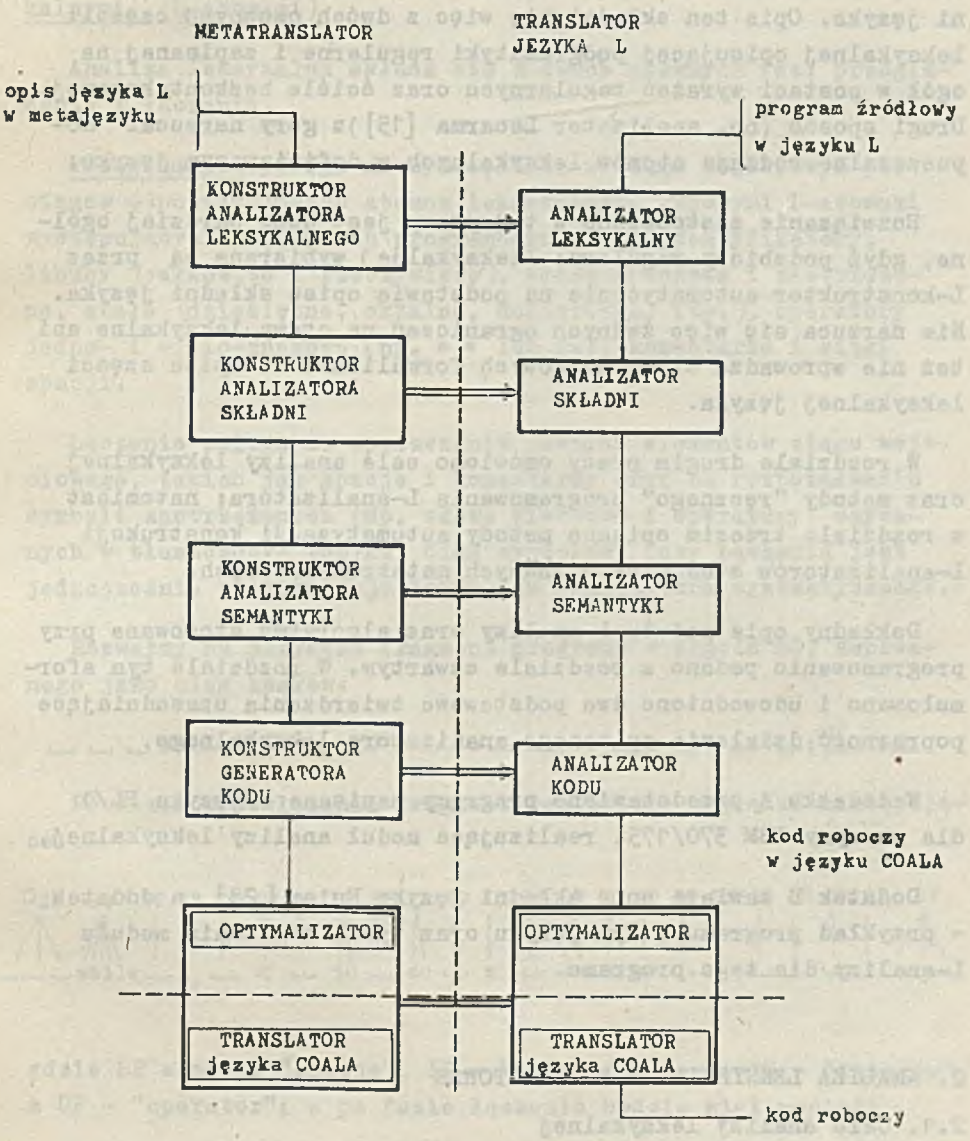
Praca niniejsza dotyczy właśnie problemów związanych z automatyczną konstrukcją analizatorów leksykalnych w metatranslatorach.

Ogólny schemat metatranslatora (ang. compiler-compiler) realizowanego w Zakładzie Teorii Translatorów Instytutu Maszyn Matematycznych przedstawia rysunek 1.

Schemat ten jest realizowany przez pakiet modułów, zwanych odpowiednio: modułem analizy leksykalnej, analizy syntaktycznej, analizy semantycznej, itd.

Głównym wynikiem pracy jest zaprogramowanie, opisanie i uzasadnianie modułu L-analizy, na który składają się programy ładowacza, L-konstruktora i L-analizy.

Dotychczas stosowane dwa sposoby rozwiązania problemu analizy leksykalnej w metatraslatorach. Pierwszy (np. AED-RWORD


METATRANSLATOR
TRANSLATOR JĘZYKA L
opis języka L w metajęzyku
program źródłowy w języku L
KONSTRUKTOR ANALIZATORA LEKSYKALNEGO
ANALIZATOR LEKSYKALNY
KONSTRUKTOR ANALIZATORA SKŁADNI
ANALIZATOR SKŁADNI
KONSTRUKTOR ANALIZATORA SEMANTYKI
ANALIZATOR SEMANTYKI
KONSTRUKTOR GENERATORA KODU
ANALIZATOR KODU
kod roboczy w języku COALA
OPTYMALIZATOR
OPTYMALIZATOR
TRANSLATOR języka COALA
TRANSLATOR języka COALA
kod roboczy

Rys. 1.

[11]) polegał na wydzieleniu części leksykalnej w opisie składni języka. Opis ten składał się więc z dwóch osobnych części: leksykalnej opisującej podgramatyki regularne i zapisanej na ogół w postaci wyrażeń regularnych oraz ściśle bezkontekstowej. Drugi sposób (np. analizator Lecarma [15])z góry narzucał dopuszczalne rodzaje atomów leksykalnych w definiowanym języku.

Rozwiązanie zastosowane w tej pracy jest dużo bardziej ogólne, gdyż podzbiory regularne (leksykalne) wybierane są przez L-konstruktor automatycznie na podstawie opisu składni języka. Nie narzuca się więc żadnych ograniczeń na atomy leksykalne ani też nie wprowadza się dodatkowych formalizmów do opisu części leksykalnej języka.

W rozdziale drugim pracy omówiono cele analizy leksykalnej oraz metody "ręcznego" programowania L-analizatora; natomiast w rozdziale trzecim opisano metody automatyzacji konstrukcji L-analizatorów stosowane w znanych metatranslatorach.

Dokładny opis modułu L-analizy oraz algorytmy stosowane przy programowaniu podano w rozdziale czwartym. W rozdziale tym sformułowano i udowodniono dwa podstawowe twierdzenia uzasadniające poprawność działania opisanego analizatora leksykalnego.

W dodatku A przedstawiono programy napisane w języku PL/1 dla maszyny IBM 370/175, realizujące moduł analizy leksykalnej.

Dodatek B zawiera opis składni języka Euler [28], a dodatek C - przykład programu w tym języku oraz wynik działania modułu L-analizy dla tego programu.

2. ANALIZA LEKSYKALNA W TRANSLATORZE

2.1. Cele analizy leksykalnej

Na pierwszym etapie procesu tłumaczenia program źródłowy ma postać ciągu znaków. Analizator leksykalny (będący pierwszym przebiegiem translatora) czyta tekst programu źródłowego rozumiany jako ciąg znaków i grupom znaków przyporządkowuje pewne

elementarne jednostki syntaktyczne, zwane dalej atomami leksykalnymi (L-atomami).

Analiza leksykalna składa się z dwóch głównych faz: przeglądania i łączenia.

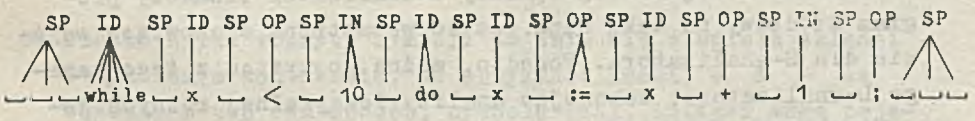
Przeglądanie polega na wydzielaniu w ciągu wejściowym podciągów odpowiadających atomom leksykalnym. Typowymi L-atomami występującymi w językach programowania są: identyfikatory, liczby (całkowite i rzeczywiste), słowa kluczowe i zastrzeżone, stałe (dziesiętne, oktalne, Holleritha, itp.), operatory jedno- i wielo-znakowe (np. * * lub :=), komentarze i ciągi spacji.

Łączenie polega na opuszczaniu pewnych elementów ciągu wejściowego, takich jak spacje i komentarze oraz na rozpoznawaniu symboli zastrzeżonych (np. słowa kluczowe i operatory) używanych w tłumaczonym języku. Ciąg wyjściowy fazy łączenia jest jednocześnie ciągiem wejściowym dla analizatora syntaktycznego.

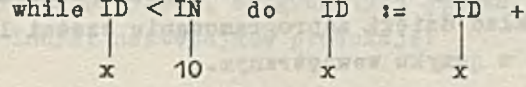
Rozważmy na przykład fragment programu w Algolu 60, zapisanego jako ciąg znaków:

␣ ␣ ␣ while ␣ x ␣ < ␣ 10 ␣ do ␣ x ␣ := ␣ x ␣ + ␣ 1 ␣ ; ␣ ␣ ␣

Po fazie przeglądania fragment ten będzie wyglądał następująco:



gdzie SP oznacza "spację", ID - "identyfikator", IN - "integer", a OP - "operator"; a po fazie łączenia będzie miał postać:



Analizator syntaktyczny otrzymuje na wejściu tylko ciąg symboli umieszczony w górnym rzędzie na powyższym rysunku, ale dodatkowe informacje (np. jakiemu konkretnemu identyfikatorowi odpowiada dany symbol ID) są przechowywane w tablicach dla przebiegu analizy semantycznej.

Redukcja tekstu wejściowego do ciągu atomów leksykalnych, które są traktowane przez S-analizator jako symbole terminalne, jest głównym celem analizy leksykalnej. Jednakże istnieje jeszcze kilka innych przyczyn wyodrębniania w translatorze przebiegu analizy leksykalnej:

1. Program źródłowy może być zapisywany za pomocą różnych urządzeń zewnętrznych, o rozmaitych zbiorach znaków. Może się też zdarzyć, że część programu znajduje się w pamięci maszyny, a pozostała jego część jest wczytywana z urządzeń zewnętrznych. Bardzo często dopuszczalne są również różne reprezentacje tego samego języka. Na przykład w niektórych reprezentacjach Algolu 60 takie ograniczniki jak begin, else, if, end, itp. występują w programie ujęte w apostrofy, spacje zaś są ignorowane, natomiast w innych reprezentacjach - ograniczniki te są traktowane jako słowa zastrzeżone, a spacje są używane jako separatory. W przypadku wyodrębnienia analizy leksykalnej możemy napisać jeden S-analizator oraz kilka L-analizatorów - znacznie prostszych i łatwiejszych do zaprogramowania - po jednym dla każdej realizacji języka i urządzenia wejściowego. Każdy L-analizator tłumaczy program źródłowy na taką samą postać wewnętrzną stanowiącą wejście dla S-analizatora. Ponadto, można korzystać z tego samego L-analizatora, dokonując analizy leksykalnej różnych języków programowania.

2. Znaczną część czasu maszyny przeznaczonego na tłumaczenie programu zajmuje badanie poszczególnych jego znaków. Wyodrębnienie analizy leksykalnej umożliwia przyspieszenie tego procesu, na przykład dzięki zaprogramowaniu części lub całego L-analizatora w języku wewnętrznym.

3. Skrócenie tekstu programu źródłowego przez L-analizator pozwala zaoszczędzić zarówno miejsca pamięci jak i czas maszyny. Jest to bardzo istotne w przypadku translatorów wieloprzebiegowych dla małych maszyn cyfrowych, gdzie program jest wielokrotnie przesyłany między pamięcią operacyjną i pomocniczą.

4. Pewne zależności kontekstowe mogą być łatwo wykryte na etapie analizy leksykalnej. Na przykład Algol 60 mógłby być zdefiniowany przez gramatykę z pierwszeństwem, gdyby nie dwuznaczne wystąpienia symbolu ":" jako ogranicznika etykiety lub w deklaracjach tablic. Na etapie analizy leksykalnej można łatwo zastąpić dwukropek dwoma różnymi symbolami - zależnie od miejsca występowania w programie, a wobec tego w S-analizatorze Algolu 60 można zastosować metody rozbioru przewidziane dla gramatyki z pierwszeństwem.

5. L-analizator może wyszukiwać (lub usuwać) niektóre trywialne błędy w programie (na przykład błędy w stałych, brak odpowiedniości między lewymi i prawymi nawiasami, itp.). Znaczne przyspieszenie translacji można osiągnąć przez usuwanie na etapie analizy leksykalnej zbędnych spacji, znaków kontynuacji, komentarzy, itp. lub przez klasyfikowanie instrukcji według typów (dla Fortranu IV ANSI Standard ta ostatnia operacja może być wykonywana przez automat skończenie stanowy).

6. Często kilka różnych symboli pojawia się w opisie składni w tym samym kontekście. Na przykład: znaki + i - są używane w ten sam sposób, symbole real, integer mogą pojawić się w tym samym miejscu programu w Algolu 60. L-analizator może przydzielić jeden symbol dla każdej grupy takich syntaktycznie synonimicznych symboli.

Załóżmy, na przykład, że gramatyka języka źródłowego zawiera miedzy innymi następujące produkcje:

```
<type>        ::= real | integer | boolean
<declaration> ::= <type> <identifier list>
<expression>  ::= <term> | <expression> + <term> | <expression>
                  - <term>
```

Gramatyka ta będzie miała po L-analizie prostszą postać, mianowicie:

<declaration> ::= TYPE <identifier list>
<expression> ::= <term> | <expression> AO <term>

gdzie TYPE oraz AO będą traktowane przez S-analizator jako symbole terminalne.

2.2. Realizacja analizatora leksykalnego

Jak już zaznaczyliśmy, składnia atomów leksykalnych jest bardzo prosta i może być zapisana w postaci gramatyki regularnej. Dlatego też analizator leksykalny będzie na ogół automatem skończenie stanowym akceptującym słowa należące do tej gramatyki. Automat ten będziemy zapisywać w postaci tzw. macierzy przejść, to jest dwuwymiarowej tablicy, której wiersze będą określały stany automatu, kolumny zaś - symbole pojawiające się na wejściu. Elementami tablicy będą wartości funkcji przejścia i funkcji wyjścia automatu.

L-analizator może być zrealizowany albo jako oddzielny przebieg, który wykonuje analizę leksykalną całego programu źródłowego i dostarcza S-analizatorowi tablicę zawierającą ten program w postaci wewnętrznej, albo jako podprogram wywoływany przez S-analizator w celu otrzymania nowego symbolu potrzebnego do analizy. W poniższym przykładzie (w którym zakładamy ten drugi wariant) korzystać będziemy z dwóch zmiennych roboczych: ZNAK, w której będziemy przechowywać aktualnie analizowany znak ciągu wejściowego, oraz ATOM, w której będziemy kompletować atom leksykalny. Przed wywołaniem analizatora leksykalnego wartość zmiennej ATOM będzie zerowana przez S-analizator.

Przykład 2.1. Rozważmy język zdefiniowany następującą gramatyką:

```
<program>      ::= .begin. <block> .end.
<block>        ::= <statement> ; | <block> <statement> ;
<statement>    ::= <leftside> <number>
<leftside>     ::= <identifier> :=
<identifier>   ::= <letter> | <identifier> <letter> | <identi-
                   fier> <digit>
<number>       ::= <integer> | <real>
<integer>      ::= <digit> | <integer> <digit>
<real>         ::= <integer> . <integer>
<letter>       ::= a | b | d | e | g | i | n
<digit>        ::= 0 | 1
```

W języku tym występują następujące atomy leksykalne:

1. operator - znak ":=" lub ";"
2. liczba całkowita - dowolny ciąg zer i jedynek (np. 101001)
3. liczba rzeczywista - atomy te są utworzone z części całkowitej i ułamkowej. Część całkowita jest ciągiem zer i jedynek; część ułamkowa ciągiem zer i jedynek poprzedzonym kropką (np. 10.011)
4. identyfikator - dowolny ciąg liter (a, b, d, e, g, i, n) i cyfr (0, 1) zaczynający się od litery (np. a10)
5. słowo kluczowe - identyfikator ograniczony z obu stron kropkami (np. .begin.).

Macierz przejść dla naszego języka jest przedstawiona w poniższej tablicy:

STAN \ SYMBOL WEJŚCIOWY	a b d e g i n		0|1	kropka	spacja, nowa linia	operator := lub;	inny znak
	f.przej.	f.wyj.					
1 (początkowy)	2	CONT	3,CONT	7,CONT	1,CONT1	1,END1	-,ERR
2 (identyfikator)	2	CONT	2,CONT	2,END	2,END	2,END	-,ERR
3 (l. całkowita)	3	END	3,CONT	6,CONT	3,END	3,END	-,ERR
4 (l. rzeczywista)	4	END	4,CONT	4,END	4,END	4,END	-,ERR
5 (słowo kluczowe)	5	CONT	5,CONT	5,END1	-,ERR	-,ERR	-,ERR
6	-	ERR	4,CONT	-,ERR	-,ERR	-,ERR	-,ERR
7	5	CONT	-,ERR	-,ERR	-,ERR	-,ERR	-,ERR

funkcje wyjścia realizują następujące zadania*):

CONT: ATOM←—ATOM || ZNAK;
Wykonaj funkcję CONT1.

CONT1: ZNAK←—kolejny znak ciągu wejściowego;
analizuj dalej tablicę.

END1: ZNAK←—kolejny znak ciągu wejściowego;
wykonaj funkcję END.

END: Załaduj ATOM do słownika symboli;
wróć do S-analizatora podając adres ATOMU w słowniku symboli i zaznaczając, że ATOM jest:

- operatorem - jeżeli stan = 1,
- identyfikatorem - jeżeli stan = 2,

*) Znak ←— oznacza operację przypisania, a znak || operację konkatenacji

- liczbą całkowitą - jeżeli stan = 3,
- liczbą rzeczywistą - jeżeli stan = 4,
- słowem kluczowym - jeżeli stan = 5.

ERR: Wróć do analizatora syntaktycznego sygnalizując błąd.

Przyporządkujmy poszczególnym atomom leksykalnym następujące numery, stanowiące ich umowną reprezentację:

nie zdefiniowany : 0,
operator : 1,
identyfikator : 2,
liczba całkowita : 3,
liczba rzeczywista : 4,
słowo kluczowe : 5.

Przykład 2.2. Rozpatrzmy następujący program w naszym języku:

.begin. ab1 := 11.01; ab2 := 11.01; .end.

etapy pracy L-analizatora wyglądają następująco:

etap	wyjście	słownik symboli						
		1	2	3	4	5	6	7
1	5,1	.begin.	-	-	-	-	-	-
2	2,2	.begin.	ab1	-	-	-	-	-
3	1,3	.begin.	ab1	:=	-	-	-	-
4	4,4	.begin.	ab1	:=	11.01	-	-	-
5	1,5	.begin.	ab1	:=	11.01	;	-	-
6	2,6	.begin.	ab1	:=	11.01	;	ab2	-
7	1,3	.begin.	ab1	:=	11,01	;	ab2	-
8	4,4	.begin.	ab1	:=	11.01	;	ab2	-
9	1,5	.begin.	ab1	:=	11.01	;	ab2	-
10	5,7	.begin.	ab1	:=	11.01	;	ab2	.end.

Na każdym etapie L-analizator generuje na wyjściu dwie wartości, z których pierwsza oznacza umowną reprezentację atomu leksykalnego, a druga adres, pod którym dany atom jest pamiętany w słowniku symboli.

3. PRZEGLĄD METOD AUTOMATYCZNEJ KONSTRUKCJI ANALIZATORÓW LEKSYKALNYCH W ZNANYCH METATRANSLATORACH

3.1. Najprostsze rozwiązania problemów automatyzacji konstrukcji L-analizatorów

Większość istniejących metatranslatorów w ogóle nie generuje przebiegu analizy leksykalnej (BMCC [22], Ingerman [10], TMG [18]); w niektórych zaś sprowadza się go do wykonywania kilku zupełnie trywialnych operacji.

Na przykład w Cogencie [21] można podawać opis znaków, tzn. dowolnie definiować kod reprezentantów znaków źródłowych; można też wprowadzać do definicji składni stałe teksty (np. słowa zastrzeżone języka), które przed S-analizatorem przetwarza analizator leksykalny i zamienia na określony układ bitów. Dokonuje się tego za pomocą definicji znaków, które określają wartości poszczególnych reprezentantów, wyrażone liczbami ósemkowymi.

Opis znaków w Cogencie ma następującą postać:

```
<opis znaków>                ::= <puste> Ø CHARDEF <ciąg definicji
                                 znaków>
<ciąg definicji znaków>      ::= <puste>| <ciąg definicji znaków>
                                 <definicja znaku>
<definicja znaku>            ::= <reprezentant znaku źródłowego> =
                                 <ciąg kodów źródłowych> <ciąg ko-
                                 dów nieistotnych> <kod wynikowy>
<ciąg kodów źródłowych>      ::= (<ciąg kodów>)
<ciąg kodów nieistotnych>    ::= <puste>| (<ciąg kodów>)
<kod wynikowy>               ::= <dodatnia całkowita ósemkowa>
<ciąg kodów>                 ::= <dodatnia całkowita ósemkowa>|
                                 <ciąg kodów>,<dodatnia całkowi-
                                 ta ósemkowa>
<reprezentant znaku źródłowego> ::= <normalny znak>|(<spe-
                                 cjalny znak>)|(Ø<tekst nazwowy>)
```

Każda definicja znaku określa wartości reprezentanta występującego po lewej stronie znaku równości. Przede wszystkim zadaje ona jedną lub kilka wartości ósemkowych, które ten repre-

zentant może przyjmować na wejściu. Na przykład, gdyby opis języka mógł wchodzić do maszyny w kodach M2, BCDIC oraz KW8*), wówczas w przypadku definicji litery A, składowa <ciąg kodów źródłowych> miałaby postać: (30, 301, 202).

Składowa < ciąg kodów nieistotnych > określa kodowe wartości znaków, które należy opuścić przy wczytywaniu, jeżeli występują one za definiowanym znakiem. Na przykład, dla kodu KW8 następująca definicja spacji (niepełna - bez określenia składowej <kod wynikowy>)

$ SP = (100) (100,24)

spowoduje, że po wczytaniu jednej spacji jako znaku z wejścia, następujące za nią spacje oraz nowe linie będą ignorowane.

Składowa <kod wynikowy > określa wartość kodową opisanego znaku na wyjściu konstruowanego kompilatora. Zakładając na przykład, że mamy do czynienia z kodem KW8 zarówno na wejściu, jak i na wyjściu, możemy znak ";" zdefiniować następująco:

$ CL = (164) 164

Zbiór znaków metajęzyka Cogent ma (w realizacji na CDC 3600) zadane z góry standardowe definicje. Dlatego w opisie języka źródłowego, sformułowanym w metajęzyku Cogent, definiujemy jedynie te znaki, które do tego metajęzyka nie należą oraz np. słowa zastrzeżone. Początek opisu języka może mieć następującą postać (przyjmując kod KW8):

```
$ CHARDEF $ CL = (164)164 $ SE = (166)166
          $ BEGIN = (302)0 $ END = (303)0
          $ REAL = (304)0 $ GOTO = (307)0
```

Trochę bardziej rozbudowany aparat do analizy leksykalnej istnieje w języku Trandir [20]. Rolę analizy leksykalnej pełni w tym języku specjalna procedura SCAN. Wywołanie SCAN(n) powoduje wczytanie n słów z ciągu wejściowego i umieszczenie ich

*) Przez kod rozumiemy przypisanie każdemu znakowi pewnej liczby. Na przykład: litera A ma w kodzie M2 wartość: 30, w kodzie BCDIC: 301, a w kodzie KW8: 202

w odpowiednich tablicach. Projektant języka definiuje tablice słów (ich pola i wymiary) za pomocą odpowiednich reguł języka Trandir. Jednocześnie może on definiować klasy słów syntaktycznie synonimicznych oraz równoważne postacie zewnętrzne dla tych samych symboli. Część programu w języku Trandir może wyglądać następująco:

```
   DECLARE TABLES              : deklaracja tablic
    OPTAB  (1.25 )             : słownik operacji języka o wymia-
    WITH FIELDS                  rze 25 oraz o dwóch polach
         COPLN
         OPCLASS
   END
   DECLARE ENTRIES
    OPTAB FORMAT               : definicja elementów tablicy
    $ DECLARE..
    $ INT..
    $ GO..
    $ EQ..          EQUAL      RELOP
    $ NE..          NEQUAL     RELOP
    $ GT..          GREAT      RELOP
    $ GE..          GREATEQ    RELOP
    $ LT..          LESS       RELOP
    $ LE..          LESSEQ     RELOP
    $ END..
      +..           PLUS       ADOP
      -..           MINUS      ADOP
END
                      ↑          ↑
                (pole COPLN)  (pole OPCLASS)
```

W tablicy OPTAB są zgromadzone operatory języka wejściowego, ich odpowiedniki w języku przejściowym (pole COPLN) oraz klasy równoważności do celów analizy syntaktycznej (pole OPCLASS).

Jeszcze inne możliwości zastosowania analizy leksykalnej zapewnia metatranslator ATWS [24]. W systemie tym projektant języka definiuje atomy leksykalne oraz procedury do rozpozna-

wania tych atomów. Na podstawie tych definicji konstruowany jest L-analizator służący do wydzielania L-atomów w programie źródłowym.

Przykład programu w języku ATWS:

```
begin
.....
class ELEM, LETTER, DIGIT, TOKEN, KEYWORD,..., SYMBOL: lista
                                        atomów leksykalnych
phase <identifier> = begin                          : defini-
                     <identifier  ::=       cja procedury rozpoz-
                          <l>{<ld>}         nającej
                     <ld>          ::= exclusive {<l>}{<d>}
                     <l>           ::= LETTER
                     <d>           ::= DIGIT
                      end
phase  <keyword>  = ...
phase  <string>   = ...
init    ELEM  with (,...,',      : definicja L-atomów
        LETTER with  (a, b,..., z),
        .
        .
        .
        ID    with <identifier> ,
        KEYWORD with <keyword> ,
        ...
        SYMBOL  with  (begin, end, int, +, ...)
```

3.2. System AED RWORD

Omówione powyżej systemy albo nie są zrealizowane w praktyce (ATWS), albo nie są metatranslatorami lecz językami do pisania translatorów, w związku z czym L-analizator musi być programowany "ręcznie" przez programistę, nie zaś generowany automatycznie na podstawie zadanej gramatyki. Jedynymi działającymi obecnie rozwiązaniami automatycznej konstrukcji L-analizatora są: system AED RWORD [11] oraz L-analizator opisany w pracy [15], zwany poniżej L-analizatorem Lecarma.

System AED RWORD (Read a WORD) jest pierwszym systemem, w którym L-analizator jest generowany automatycznie. Jest on częścią systemu AED-1, którego celem jest generowanie kompilatorów, interpretatorów, systemów operacyjnych, itd. System AED RWORD jest stosowany nie tylko do celów analizy leksykalnej. Uniwersalność systemu powoduje, że jest on mniej efektywny. L-analizator skonstruowany przez RWORD na maszynę IBM 7094 przetwarzał 3000 znaków/s, podczas gdy L-analizator zakodowany "ręcznie" przetwarzał 8000 znaków/s.

W systemie RWORD zakłada się wydzielenie w opisie składni języka podgramatyk regularnych, na podstawie których konstruowane są automaty skończone. Podgramatyki te stanowią dane wejściowe do systemu i są zapisywane w następującej postaci:*/

BEGIN { <opis klasy znaków> } END
BEGIN { <opis atomu leksykalnego> } END
FINI

3.2.1. Opis klasy znaków

Klasa znaków może być zdefiniowana dwoma sposobami: w pierwszym znaki wymienia się explicite, w drugim - podaje się ich wewnętrzną reprezentację (w kodzie EBCDIC). Na przykład:

```
LET = /ABC... YZ/
PUNCTUATION = A+-.,; =/A
XX = ¢21,22,25¢
```

Pierwszy znak następujący po znaku "=" i różny od spacji jest ogranicznikiem klasy znaków. Użycie znaku "¢" jako ogranicznika oznacza, że definiujemy klasę przez podanie reprezentacji znaków w kodzie EBCDIC. Klasa LET składa się więc z wszystkich liter, a klasa PUNCTUATION - ze znaków + - . , ; = oraz /. Klasa XX składa się ze znaków o reprezentacji wewnętrznej 21, 22, 25.

*) Zapis {A} oznacza ciąg (być może pusty) powtórzeń symbolu A

Nazwa klasy musi być różna od BEGIN, END, FINI. Nazwa IGNORE jest używana do zdefiniowania klasy znaków ignorowanych przez L-analizator.

3.2.2. Opis atomu leksykalnego

Opis atomu leksykalnego ma następującą postać:

<identyfikator> (<integer>)' = < wyrażenie regularne > $

gdzie zmiennymi wyrażenia regularnego mogą być:

a) dowolne znaki kodu EBCDIC różne od |, (,) ., ´ , $ i znaku spacji,
b) puste słowo ε,
c) nazwy klas znaków,
d) znak apostrofu z następującym po nim dowolnym znakiem kodu EBCDIC różnym od spacji; wartością takiej pary jest znak następujący po apostrofie (jest to jedyna możliwość używania znaków niedopuszczalnych w punkcie a).

Przykłady:

PUNCT (5) = PUNCTUATION $

Klasa leksykalnych atomów PUNCT składa się ze znaków z klasy PUNCTUATION; atomy z tej klasy mają umowną reprezentację 5.

BEG (3) = ´´BEGIN´´ $

oznacza, że słowo ´BEGIN´ ma umowną reprezentację 3,

ID (1) = LET (LET | DIGIT)* 5 $

oznacza, że słowo złożone najwyżej z sześciu liter i cyfr, którego pierwszym znakiem jest litera, ma umowną reprezentację 1.

Specjalną klasą jest klasa IGNORE. Każde słowo z tej klasy jest ignorowane. Ponieważ umowna reprezentacja nie jest w tym przypadku potrzebna, definicja tej klasy ma następującą postać:

IGNORE = < wyrażenie regularne > $

Zauważmy różnicę między klasą znaków IGNORE i klasą atomów leksykalnych IGNORE. Na przykład:

```
BEGIN   IGNORE = / /          LET = /AB/      END
BEGIN   ID (1) = LET (LET)* $                 END
FINI
```

Ciąg znaków AB A będzie traktowany jako identyfikator ABA, ponieważ znak spacji jest ignorowany. Natomiast przy następującej definicji:

```
BEGIN  SPACE = /  /      LET = /AB/          END
BEGIN  IGNORE = SPACE $  ID  = LET (LET)* $  END
FINI
```

ciąg AB A będzie traktowany jako dwa identyfikatory AB i A.

Klasę atomów leksykalnych można również zadeklarować w następujący sposób:

<identyfikator>(<integer> , <sub name>) = <wyrażenie regularne> $

wtedy za każdym razem po skonstruowaniu atomu należącego do klasy <identyfikator> zostaje wywołana procedura <sub name>. Procedura związana z identyfikatorem może umieszczać go w słowniku symboli i sprawdzać, czy dany identyfikator jest słowem zastrzeżonym, czy też nie.

W wielu przypadkach można dokonać rozbioru wejściowego ciągu znaków kilkoma sposobami. System RWORD wykrywa wszystkie wieloznaczności i podejmuje jedną z następujących akcji:

1. Podejmuje decyzję usunięcia wieloznaczności. Na przykład ciąg znaków AB może oznaczać zarówno identyfikator AB jak i dwa identyfikatory A oraz B. System RWORD konstruuje wtedy najdłuższe możliwe słowo i przechodzi do dalszej analizy.

2. Podejmuje decyzję usunięcia wieloznaczności i powiadamia o tym operatora. Na przykład ciąg znaków X123 może być traktowany jako identyfikator X123 lub też jako identyfikator X oraz liczba całkowita 123. W takim przypadku RWORD też konstruuje najdłuższe słowo.

3. Komunikuje, że wieloznaczność jest zbyt skomplikowana i wymaga analizy kontekstu (na przykład instrukcja Fortranu DO10I = 1.20, której pierwsza część (DO10I) mogłaby zostać potraktowana jako identyfikator), po czym kończy pracę.

Poniżej pokażemy, jak można by zapisać jako dane wejściowe systemu RWORD część języka z przykładu 2.1.

```
BEGIN  SPACE = /  /
       LET   = /abdegin/
       DIG   = /01/
       OP    = /;:=/
       DOT   = /./
END
BEGIN  OPR(1) = OP$
       ID (2,LOOKUP) = LET (LET| DIG)* $
       INT(3,CONVERTI) = DIG (DIG)* $
       REAL (4,CONVERTR) = DIG (DIG)* DOT DIG (DIG)* $
       KEY (5) = (.begin.|.end.) $
END    FINI
```

Procedura LOOKUP będzie umieszczała identyfikator w słowniku symboli, a procedury CONVERTI oraz CONVERTR obliczały wartość numeryczną liczby.

W pewnych językach programowania można wydzielić kilka podjęzyków, w których mogą być zupełnie inne reguły tworzenia słów (np. w Algolu 60 spacje mają znaczenie wewnątrz tekstów, a poza nimi są ignorowane). W takim przypadku dobrze jest mieć kilka różnych L-analizatorów (po jednym dla każdego podjęzyka) oraz aparat pozwalający przechodzić od jednego L-analizatora do drugiego. Na przykład, gdyby w powyższym języku

założyć, że komentarz zaczyna się i kończy znakiem "/", wówczas można by wprowadzić definicję:

```
SL '6,COMMENT' = '/$
```

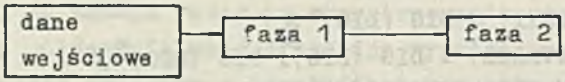
gdzie procedura COMMENT wywołuje inny, podany poniżej L-analizator, który ominie komentarz i powróci do poprzedniego L-analizatora.

L-analizator akceptujący komentarz wygląda następująco:

```
BEGIN IGNCHAR = $00,01,...,77$ 'wszystkie znaki poza "/"
END
BEGIN IGNORE = IGNCHAR $
TERMIN (1,ENDCOMMENT) = '/$
END  FINI
```

gdzie procedura ENDCOMMENT przekazuje sterowanie do głównego L-analizatora.

System RWORD składa się z dwóch części. Pierwszą z nich można zilustrować następującym diagramem:



W fazie 1 są tworzone automaty niedeterministyczne dla każdego wyrażenia regularnego. W fazie 2 automaty te są łączone i powstaje jeden automat deterministyczny zapisany w języku AED-0. Składa się on z wywołań procedur i tablic skonstruowanych w obu fazach.

Zadaniem drugiej części systemu RWORD jest optymalizacja wyprodukowanego programu ze względu na czas wykonywania i wykorzystanie pamięci maszyny.

3.3. L-analizator Lecarma

W systemie Lecarma przyjęto jako główne założenie fakt, że w większości języków programowania powtarzają się typowe atomy

leksykalne. Wobec tego w systemie tym zdefiniowano następujące L-atomy:

a) identyfikator – ciąg liter i cyfr zaczynający się od litery; długość maksymalna identyfikatora jest ustalana przez projektanta języka;

b) słowo zastrzeżone – atom o analogicznej składni jak identyfikator, o ustalonym z góry znaczeniu;

c) słowo kluczowe – ciąg liter i cyfr zaczynający się od litery i ograniczony z obu stron specjalnym znakiem zdefiniowanym przez projektanta języka;

d) operator – jeden lub dwa znaki różne od litery, cyfry i spacji;

e) liczba całkowita – ciąg cyfr dziesiętnych (bez znaku);

f) liczba rzeczywista – atomy te są utworzone z części całkowitej, części ułamkowej i części wykładniczej, przy czym druga lub trzecia część może nie występować;
część całkowita jest ciągiem cyfr dziesiętnych;
część ułamkowa składa się z ciągu cyfr dziesiętnych następującego po znaku DELFRACTION - definiowanym przez projektanta (domyślnie przyjmuje się kropkę); część wykładnicza jest oznaczona znakiem DELEXPOSAN - również definiowanym przez projektanta (domyślnie litera E) - po którym następuje ciąg cyfr poprzedzony ewentualnie znakiem + lub - ;

g) tekst – ciąg dowolnych znaków ograniczony z lewej strony przez znak DEBCHAINE, a z prawej strony przez znak FINCHAINE;

h) komentarz – ciąg dowolnych znaków ograniczony z lewej strony przez znak DEBCOMMENT, a z prawej strony przez znak FINCOMMENT.

Projektant języka definiuje tylko pewne uzupełnienia do powyższych definicji, jak np.:

DEBCHAINE, FINCHAINE – znaki początku i końca tekstu,
DEBCOMMENT, FINCOMMENT – znaki początku i końca komentarza,
DELEXPOSANT, DELFRACTION – znaki reprezentujące odpowiednio wykładnik i kropkę dziesiętną w liczbie rzeczywistej,
DELMOTCLÉ – znak ograniczający słowo kluczowe (z obu stron),
MAXIDENT – maksymalna dopuszczalna długość identyfikatora.

W związku z tym automaty akceptujące atomy leksykalne są zaprogramowane "z góry" i wprowadza się do nich - dla konkretnego języka - tylko nieznaczne poprawki, co umożliwia bardzo efektywną pracę L-analizatora.

Wadą tej metody jest fakt ograniczenia liczby atomów leksykalnych, jednakże autorzy twierdzą, że w praktyce bardzo rzadko spotyka się języki mające inne L-atomy.

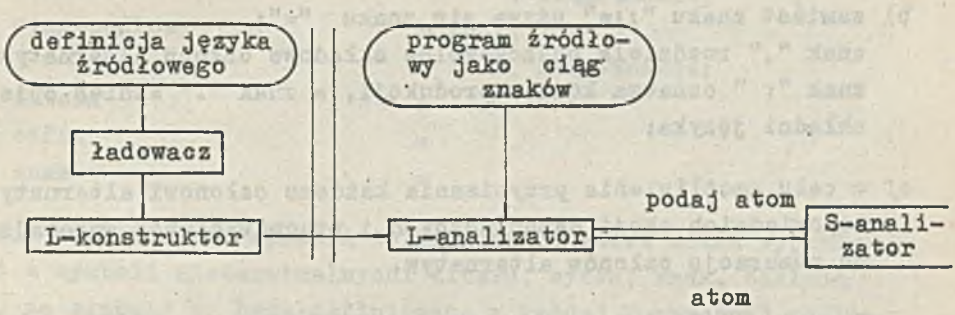
4. OPIS KONSTRUKCJI MODUŁU ANALIZY LEKSYKALNEJ

Przedstawimy w tym rozdziale nowe rozwiązanie koncepcji L-analizatora, zastosowane w metatranslatorze realizowanym w Instytucie Maszyn Matematycznych. Proponowana metoda polega na automatycznym wydzielaniu podzbiorów regularnych języka, dokonywanym na podstawie opisu jego składni. W związku z tym nie ma potrzeby wprowadzania dodatkowego metajęzyka, służącego do opisu części leksykalnej (co jest najpoważniejszą wadą systemu

RWORD), Nie narzuca się również (w odróżnieniu od L-analizatora Lecarma) żadnych ograniczeń na postać atomów leksykalnych.

4.1. Organizacja ogólna

Moduł analizy leksykalnej składa się z trzech podmodułów: ładowacza, L-konstruktora i L-analizatora. Ogólny schemat działania tych podmodułów wygląda następująco:



Wszystkie podmoduły zostały zaprogramowane w języku PL/1 dla maszyny IBM 370/175. Podmoduły ładowacza i L-analizatora są realizowane przez procedury o nazwach ŁADOWACZ I LANALIZATOR, natomiast na podmoduł L-konstruktora składają się dwie procedury - LKONSTRUKTOR oraz REGULAR. Całość programu zawiera około 800 instrukcji w języku PL/1 i zajmuje w pamięci maszyny około 22000 bajtów.

W dodatku A zamieszczono teksty wszystkich programów.

4.2. Podmoduł ładowacza

Zadaniem podmodułu jest wczytanie opisu składni języka źródłowego, umieszczenie jej w pamięci maszyny oraz utworzenie pewnych tablic, z których będą korzystały kolejne podmoduły. Realizuje to procedura zewnętrzna - ŁADOWACZ, zawierająca około 200 instrukcji w języku PL/1.

4.2.1. Metajęzyk opisu składni języka źródłowego

Metajęzyk opisu składni oparty jest w zasadzie na metajęzyku Backusa-Naura, do którego wprowadzono następujące zmiany:

a) w celu umożliwienia formalnego zdefiniowania metajęzyka opisu składni w tym metajęzyku, wprowadzono apostrofy ograniczające symbole terminalne gramatyki; nie stosuje się natomiast nawiasów kątowych, używanych w notacji BNF do ograniczania symboli nieterminalnych;

b) zamiast znaku "::=" używa się znaku "=";
znak "," rozdziela poszczególne składowe członu alternatywy;
znak "; " oznacza koniec produkcji, a znak "." koniec opisu składni języka;

c) w celu umożliwienia przypisania każdemu członowi alternatywy odpowiednich akcji semantycznych i pragmatycznych wprowadzono numerację członów alternatyw.

Formalna definicja metajęzyka opisu składni zapisana w tym metajęzyku jest następująca:

```
1: opis-gramatyki          = ciąg-produkcji '.';
2: ciąg-produkcji          = produkcja|
3:                           produkcja, ';', ciąg-produkcji;
4: produkcja               = ident-czł-altern, ':', l-str-
                             prod, '=', p-str-prod;
5: ident-czł-altern        = cyfra|
6:                           cyfra, ident-czł-altern;
7: l-str-prod              = nieterminal;
8: nieterminal             = litera|
9:                           nieterminal, znak-1;
10: znak-1                 = litera|
11:                          cyfra|
12:                          '-';
13: p-str-prod             = alternatywa;
14: alternatywa            = ciąg-członów-altern;
15: ciąg-członów-altern    = człon-alternatywy|
```

```
16:                                  człon-alternatywy, '|', ident-
                                     czł-altern,':', ciąg-członów-
                                     altern;
17: człon-alternatywy              = ciąg-składowych-czł-altern;
18: ciąg-składowych-czł-altern     = skład-czł-altern|
                                     skład-czł-altern,',',ciąg-
                                     składowych-czł-altern;
20: skład-czł-altern               = symbol;
21: symbol                         = terminal|
22:                                  nieterminal;
23: terminal                       = '''', ciąg-znaków, '''';
24: ciąg-znaków                    = znak|
25:                                  znak, ciąg-znaków;
26: litera                         =
27: cyfra                          =
28: znak                           =
```

Nie zdefiniowano symboli terminalnych, które można wyprowadzić z symboli nieterminalnych: litera, cyfra, znak. Zakłada się, że symbole te będą definiowane w każdej konkretnej realizacji.

Przykład 4.1. Rozważmy gramatykę z przykładu 2.1 usuwając z niej tylko kropki ograniczające słowa begin i end. W naszym metajęzyku gramatyka ta będzie zapisana następująco:

```
1: program                         = 'begin', block, 'end';
2: block                           = statement, ';'|
3:                                   block, statement, ';' ;
4: statement                       = leftside, number;
5: leftside                        = identifier, ':=';
6: identifier                      = letter|
7:                                   identifier, letter|
8:                                   identifier, digit;
9: number                          = integer|
10:                                  real;
11: integer                        = digit|
12:                                  integer, digit;
13: real                           = integer, '.', integer;
```

```
14: letter      = 'a'|
15:               'b'|
16:               'd'|
17:               'e'|
18:               'g'|
19:               'i'|
20:               'n';
21: digit       = '0'|
22:               '1'.
```

4.2.2. Procedura ŁADOWACZ

Procedura ta wczytuje opis gramatyki języka źródłowego i sprawdza jego poprawność. Wykrywane i sygnalizowane są następujące błędy:

a) po lewej stronie znaku równości w produkcji występuje symbol terminalny;
b) brak znaku "=" kończącego lewą stronę produkcji;
c) brak dwukropka po identyfikatorze produkcji;
d) symbol nieterminalny nie zaczyna się od litery;
e) brak aksjomatu gramatyki.

Po stwierdzeniu poprawności opisu gramatyki, procedura ŁADOWACZ umieszcza go w odpowiednich tablicach w pamięci maszyny.

4.2.3. Postać gramatyki języka źródłowego w pamięci maszyny

Ładowacz umieszcza wczytaną gramatykę w czterech tablicach: SYMBTAB, PRODAT, PROD i ALTER, ponadto wyszukuje aksjomat gramatyki*) i umieszcza go w specjalnej zmiennej - STARTSYMB.

Tablice SYMBTAB (tablica symboli gramatyki) i PRODAT (tablica adresów produkcji) mają rozmiary równe liczbie wszystkich symboli (nieterminalnych i terminalnych) gramatyki języka wejściowego.

*) Za aksjomat przyjmujemy symbol nieterminalny nie występujący po prawej stronie żadnej produkcji

Każdy wiersz tablicy SYMTAB jest strukturą (w sensie języka PL/1) składającą się ze zmiennej znakowej SYMBOL (w zmiennej tej zapisywany jest odpowiedni symbol gramatyki) oraz zmiennej bitowej TCNT określającej, czy dany symbol jest symbolem terminalnym czy też nieterminalnym.

Tablica PRODAT zawiera dla każdego symbolu adres, pod którym są zapisane wszystkie produkcje mające jako lewą stronę dany symbol (w przypadku symbolu terminalnego adres ten jest równy 0).

Tablica PROD (tablica produkcji) jest strukturą o dwóch polach. Pierwsze z nich (ILOŚĆ) jest zmienną liczbową określającą liczbę członów alternatywy będącej prawą stroną produkcji dla danego symbolu nieterminalnego. Drugie pole (ADRALT) - jest tablicą zawierającą adresy kolejnych członów alternatywy; rozmiar tej tablicy zależy od pola ILOŚĆ.

Tablica ALTER (tablica alternatyw) jest strukturą złożoną z trzech elementów. Dwa pierwsze elementy (DALT i IDALT) są zmiennymi liczbowymi określającymi odpowiednio: długość i identyfikator członu alternatywy. Trzeci element (SYMBOL) jest tablicą o rozmiarze równym długości członu alternatywy; w kolejnych wierszach tablicy są zapisywane adresy w tablicy SYMBTAB kolejnych składowych członu alternatywy.

Przykład 4.2. Pokażemy, jak po zakończeniu działania procedury ŁADOWACZ będzie rozmieszczona w pamięci maszyny gramatyka podana w przykładzie 4.1.

	SYMBOL	TCNT
1	program	0
2	begin	1
3	block	0
4	end	1
5	statement	0
6	;	1
7	leftside	0
8	number	0
9	identifier	0
10	:=	1
11	letter	0
12	digit	0
13	integer	0
14	real	0
15	.	1
16	a	1
17	b	1
18	d	1
19	e	1
20	g	1
21	i	1
22	n	1
23	0	1
24	1	1

tablica SYMBTAB

ADR1	← adres produkcji, której lewą stroną jest symbol "program"
0	
ADR2	
0	
ADR3	
0	
ADR4	
ADR5	
ADR6	
0	
ADR7	
ADR8	
ADR9	
ADR10	
0	
0	
0	
0	
0	
0	
0	
0	
0	
0	

tablica PRODAT

liczba członów altern. — adres członu altern.

ADR1:	ILOŚĆ	ADRALT
	1	ADR11

*)

tablica PROD

długość członu altern. — identyf. członu altern. — adresy w SYMBTAB kolej.skł. członu alternat.

ADR11:	DALT	IDALT	SYMBOL		
	3	1	2	3	4

tablica ALTER

ADR2:	ILOŚĆ	ADRALT	
	2	ADR12	ADR13

tablica PROD

ADR12:	DALT	IDALT	SYMBOL	
	2	2	5	6

tablica ALTER

ADR13:	DALT	IDALT	SYMBOL		
	3	3	3	5	6

tablica ALTER

ADR3:	ILOŚĆ	ADRALT
	1	ADR14

tablica PROD

ADR14:	DALT	IDALT	SYMBOL	
	2	4	7	8

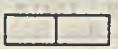
tablica ALTER

ADR4:	ILOŚĆ	ADRALT
	1	ADR15

tablica PROD

ADR15:	DALT	IDALT	SYMBOL	
	2	5	9	10

tablica ALTER

*) rysunek ADRi:  oznacza, że pierwszy element tablicy A zapamiętany jest pod adresem ADRi.

tablica A

ADR5:

ILOŚC	ADRALT	
2	ADR16	ADR17

tablica PROD

ADR16:

DALT	IDALT	SYMBOL
1	9	13

tablica ALTER

ADR17:

DALT	IDALT	SYMBOL
1	10	14

tablica ALTER

ADR6:

ILOŚĆ	ADRALT		
3	ADR18	ADR19	ADR20

tablica PROD

ADR18:

DALT	IDALT	SYMBOL
1	6	11

tablica ALTER

ADR19:

DALT	IDALT	SYMBOL	
2	7	9	11

tablica ALTER

ADR20:

DALT	IDALT	SYMBOL	
2	8	9	12

tablica ALTER

ADR7:

ILOŚĆ	ADRALT		
7	ADR21 ADR24 ADR27	ADR22 ADR25	ADR23 ADR26

tablica PROD

ADR21:

DALT	IDALT	SYMBOL
1	14	16

tablica ALTER

ADR22:

DALT	IDALT	SYMBOL
1	15	17

tablica ALTER

ADR23:

DALT	IDALT	SYMBOL
1	16	18

tablica ALTER

ADR24:

DALT	IDALT	SYMBOL
1	17	19

tablica ALTER

ADR25:

DALT	IDALT	SYMBOL
1	18	20

tablica ALTER

ADR26:

DALT	IDALT	SYMBOL
1	19	21

tablica ALTER

ADR27:

DALT	IDALT	SYMBOL
1	20	22

tablica ALTER

ADR8::

ILOŚĆ	ADRALT	
2	ADR28	ADR29

tablica PROD

ADR28:

DALT	IDALT	SYMBOL
1	21	23

tablica ALTER

ADR29:

DALT	IDALT	SYMBOL
1	22	24

tablica ALTER

ADR9:

ILOŚĆ	ADRALT	
2	ADR30	ADR31

tablica PROD

ADR30:

DALT	IDALT	SYMBOL
1	11	12

tablica ALTER

ADR31:

DALT	IDALT	SYMBOL	
2	12	13	12

tablica ALTER

ADR10:

ILOŚĆ	ADRALT
1	ADR32

tablica PROD

ADR32:

DALT	IDALT	SYMBOL		
3	13	13	15	13

tablica ALTER

4.3. Podmoduł L-konstruktora

Podmoduł ten znajduje podjęzyki regularne zadanego języka źródłowego i tworzy deterministyczny automat skończenie stanowy akceptujący te podjęzyki. Określenie podjęzyków regularnych realizuje procedura REGULAR, a tworzenie automatu - procedura LKONSTRUKTOR. Każda z procedur zawiera około 80 instrukcji w języku PL/1.

4.3.1. Procedura REGULAR

Wejściem dla procedury REGULAR są tablice SYMBTAB, PRODAT, PROD i ALTER; wyjściem - tablica RCNR. Tablica ta ma rozmiar równy rozmiarowi tablicy SYMBTAB. Każdy element tablicy RCNR przyjmuje wartość 1 lub 0 zależnie od tego, czy odpowiadający mu symbol tablicy SYMBTAB, przyjęty jako aksjomat języka, generuje język regularny czy też nie.

Algorytm działania procedury REGULAR jest następujący:

Krok 1: Dla każdego j od 1 do N (N jest rozmiarem tablicy SYMBTAB) wykonaj:

RCNR(j) ←— TCNT(j)

Krok 2: ROB ←— 0

Krok 3: Dla każdego j od 1 do N, takiego że RCNR(j) = 0 wykonaj: Oznaczmy przez A wartość pola SYMBOL w j-tym wierszu tablicy SYMBTAB, a przez P_A zbiór wszystkich członów alternatywy produkcji, której lewą stroną jest A. Jeżeli dla każdego członu p ze zbioru P_A, p jest jednej z następujących postaci: B lub CB, lub AB i przy tym RCNR(k) = 1, TCNT(l) = 0 i RCNR(l) = 1, gdzie k, l są odpowiednio adresami B, C w SYMBTAB, to wykonaj:

RCNR(j) ←— 1 oraz ROB ←— 1;

Krok 4: Jeżeli ROB = 1, to przejdź do wykonywania kroku 2, w przypadku przeciwnym zakończ pracę procedury REGULAR.

Przykład 4.3. Zilustrujemy działanie powyższego algorytmu dla gramatyki podanej w przykładzie 4.1.

Po wykonaniu kroków 1 i 2 zmienna ROB = 0, a tablica RCNR wygląda następująco:

1	0
2	1
3	0
4	1
5	0
6	1
7	0
8	0
9	0
10	1
11	0
12	0
13	0
14	0
15	1
16	1
17	1
18	1
19	1
20	1
21	1
22	1
23	1
24	1

tablica RCNR

Po wykonaniu kroku 3 otrzymamy trzy nowe zmienne "regularne", mianowicie: digit, letter, integer; tzn. zostaną wykonane operacje przypisania:

RCNR(11) ← 1; RCNR(12) ← 1; RCNR(13) ← 1; oraz ROB ← 1.

Po ponownym wykonaniu kroku 2 oraz kroku 3 dochodzi nowa zmienna "regularna" - identifier, czyli RCNR(9) ← 1. Ponieważ ROB = 1, więc znów zostaną wykonane krok 2 i krok 3; otrzymamy wówczas RCNR(7) = 1. Dopiero po czwartej iteracji będzie ROB = 0 (tzn. nie otrzymamy żadnej nowej zmiennej "regularnej"), co kończy pracę algorytmu. Na wyjściu otrzymujemy następującą tablicę RCNR:

1	0
2	1
3	0
4	1
5	0
6	1
7	1
8	0
9	1
10	1
11	1
12	1
13	1
14	0
15	1
16	1
17	1
18	1
19	1
20	1
21	1
22	1
23	1
24	1

TABLICA RCNR

4.3.2. Uzasadnienie teoretyczne procedury REGULAR

Pokażemy, że procedura REGULAR znajduje rzeczywiście podjęzyki regularne zadanego języka źródłowego.

Twierdzenie 4.1

Klasa języków regularnych jest zamknięta ze względu na operację podstawienia.

Dowód patrz [6].

Definicja 4.1. Niech $G = \langle V_N, V_T, P, S \rangle$ będzie gramatyką bezkontekstową. Oznaczmy $V = V_N \cup V_T$.

Określmy funkcję częściową $\rho : V \longrightarrow I^o$ następująco*):

1. $\rho(A) = 0 \iff A \in V_T$
2. $\rho(A) = n \iff$ n jest najmniejszą z możliwych liczb K takich, że dla każdego α takiego, że $(A, \alpha) \in P$ $\alpha = B$ lub $\alpha = CB$ lub $\alpha = AB$, gdzie $\rho(B) < K$ i $0 < \rho(C) < K$.

Funkcję ρ nazywamy rzędem regularności gramatyki G.

Definicja 4.2. Gramatyka $G = \langle V_N, V_T, P, S \rangle$ jest quasi-regularna $\iff$ dla każdego $A \in V$ zachodzi $\rho(A) \in I^o$.

Symbol A należący do V będziemy nazywać quasi-regularnym $\iff \rho(A) \in I^o$.

Symbol $A \in V$ jest regularny rzędu n $\iff \rho(A) = n$.

Twierdzenie 4.2

Jeżeli gramatyka $G = \langle V_N, V_T, P, S \rangle$ jest gramatyką quasi-regularną, to język L (G) jest regularny.

---

*) Przez I^o rozumiemy zbiór liczb całkowitych nieujemnych

Dowód:

Zauważmy najpierw, że dla każdego $A \in V$ zachodzi $\rho(A) \leq \rho(S)$.

Dowód twierdzenia przeprowadzimy przez indukcję względem rzędu regularności aksjomatu gramatyki.

Jeżeli $\rho(S) = 1$, to jedynymi produkcjami z P mającymi po lewej stronie symbol S mogą być produkcje postaci $S \longrightarrow a$ lub $S \longrightarrow Sa$, gdzie $a \in V_T$; czyli język $L(G)$ jest regularny*).

Załóżmy teraz, że twierdzenie jest prawdziwe, jeżeli $\rho(S) < n$. Niech $\rho(S) = n$, tzn. produkcje z P mające po lewej stronie S są jednej z następujących postaci:

$S \longrightarrow B$ lub $S \longrightarrow CD$ lub $S \longrightarrow SE$, gdzie $\rho(B) < n$; $\rho(D) < n$; $\rho(E) < n$ oraz $0 < \rho(C) < n$.

Dla ustalenia uwagi załóżmy, że $S \longrightarrow B$, $S \longrightarrow CD$, $S \longrightarrow SE$ są jedynymi produkcjami z P mającymi po lewej stronie S.

Rozważmy gramatykę $G' = \langle V_N', V_T', P', S' \rangle$, w której:

$V_N' = \{S, S_1\}$; $V_T' = \{B, C, D, E\}$; $S' = S$; $P' = \{S \longrightarrow B,\ S \longrightarrow SE,\ S \longrightarrow S_1D,\ S_1 \longrightarrow C\}$.

Gramatyka G' jest regularna; czyli język $L(G')$ jest regularny.

Oznaczmy: $G_A = (V_N, V_T, P, A)$ i określmy na zbiorze $V' = V_N' \cup V_T'$ funkcję f następująco:

$$f(A) = \begin{cases} L(G_A) & \text{dla } A \in V_T' \\ A & \text{dla } A \in V_N' \end{cases}$$

Języki $L(G_B)$, $L(G_C)$, $L(G_D)$, $L(G_E)$ są regularne z założenia indukcyjnego.

Ponieważ $L(G')$ jest regularny, więc z twierdzenia 4.1 otrzymujemy, że $f(L(G'))$ jest także regularny. Ale $f(L(G')) = L(G)$, czyli $L(G)$ jest językiem regularnym.

c.b.d.o.

*) Parę $(A, B) \in P$ będziemy również zapisywać w postaci $A \longrightarrow B$

Wniosek

Z algorytmu konstrukcji tablicy RCNR wynika, że dla dowolnego k, jeżeli RCNR(k) = 1, to $\rho(A) \in I^{o}$, gdzie k jest adresem A w SYMBTAB. Z tego faktu i z twierdzenia 4.2 wynika, że konstruowana przez procedurę REGULAR tablica RCNR rzeczywiście "wskazuje" symbole gramatyki generujące podjęzyki regularne.

4.3.3. Procedura LKONSTRUKTOR

Procedura LKONSTRUKTOR, korzystając z tablic utworzonych przez procedury ŁADOWACZ i REGULAR, tworzy nowe tablice: TABSTAN, TABTERM i AUTOMAT będące wejściem dla L-analizatora.

Tablice TABTERM i TABSTAN (o rozmiarach - odpowiednio ILT oraz ILST) służą do zapamiętywania symboli terminalnych oraz stanów automatu.

Tablica AUTOMAT jest tablicą dwuwymiarową o liczbie wierszy i kolumn - odpowiednio ILST oraz ILT.

Każdy element tablicy AUTOMAT składa się z trzech pól: (1) pola PRZEJSCIE, reprezentowanego zmienną liczbową i określającego funkcję przejścia, (2) pole WYJSCIE (również reprezentowanego zmienną liczbową), w którym jest zapisany identyfikator produkcji odpowiadającej napotkanemu w danym stanie symbolowi terminalnemu oraz (3) jednobitowego pola REDUKCJA, określającego czy należy wczytywać nowy znak ciągu wejściowego, czy też nie.

Chociaż podjęzyki znajdowane przez procedurę REGULAR są regularne (tw. 4.2), to jednak automat konstruowany przez LKONSTRUKTOR nie jest automatem skończenie stanowym i program obsługujący go jest dużo bardziej skomplikowany niż program obsługi automatu skończonego. Wynika to z faktu, że gramatyki podjęzyków regularnych nie muszą być regularne. Oczywiście można skonstruować automat skończony akceptujący te podjęzyki; wymagałoby to jednak zmiany niektórych produkcji gramatyki wejściowej, a w związku z tym również związanych z nimi akcji semantycznych, co w przypadku automatycznej konstrukcji translatora jest niemożliwe.

Algorytm realizowany przez procedurę LKONSTRUKTOR jest następujący:

Krok 1: TABTERM (1) ←— 'TERM1';
TABSTAN (1) ←— 'STAN1';

Krok 2: dla każdego j od 1 do N (N jest rozmiarem tablicy SYMBTAB) takiego, że RCNR(j) = 1 i TCNT(j) = 0 (tzn. j-ty symbol tablicy SYMBTAB jest quasi-regularny i nieterminalny),
wykonaj:
Oznaczmy przez A wartość pola SYMBOL w j-tym wierszu tablicy SYMBTAB, a przez P_A zbiór wszystkich produkcji gramatyki, których lewą stroną jest A. Ponieważ A jest zmienną quasi-regularną więc produkcje należące do P_A mogą być tylko jednej z czterech postaci:

(I) $A \to a$ $\quad a \in V_T$
(II) $A \to Ba$ $\quad B \in V_N;\ \rho(B) \in I^0;\ a \in V_T$
(III) $A \to B$ $\quad B \in V_N;\ \rho(B) \in I^0$
(IV) $A \to BC$ $\quad B \in V_N;\ C \in V_N;\ \rho(C) \in I^0;\ \rho(B) \in I^0;$
$\rho(B) \neq 0$

Dla każdej produkcji postaci (I) wykonaj:

AUTOMAT (1,ADR2) . PRZEJSCIE ←— ADR3*)
AUTOMAT (1,ADR2) . WYJSCIE ←— NRPROD
AUTOMAT (1,ADR2) . REDUKCJA ←— 0
gdzie ADR2 oznacza adres "a" w tablicy terminali (TABTERM),
ADR3 - adres "A" w tablicy stanów (TABSTAN),
a NRPROD - identyfikator danej produkcji $A \to a$.

Dla każdej produkcji postaci (II) wykonaj:

AUTOMAT (ADR1, ADR2) . PRZEJSCIE ←— ADR3
AUTOMAT (ADR1, ADR2) . WYJSCIE ←— NRPROD
AUTOMAT (ADR1, ADR2) . REDUKCJA ←— 1

*) Zapis AUTOMAT(i, j). PRZEJSCIE oznacza pole PRZEJSCIE elementu i, j dwuwymiarowej tablicy AUTOMAT

gdzie ADR1 jest adresem "B" w TABSTAN, ADR2 - adresem "a" w TABTERM, ADR3 - adresem "A" w TABSTAN, a NRPROD - identyfikatorem produkcji: A ⟶ Ba.

Dla każdej produkcji postaci (III) wykonaj:

AUTOMAT (ADR1,1) . PRZEJŚCIE ← ADR3
AUTOMAT (ADR1,1) . WYJSCIE ← NRPROD
AUTOMAT (ADR1,1) . REDUKCJA ← 0

gdzie ADR1, ADR3 są adresami odpowiednio "B" i "A" w TABSTAN.

Dla każdej produkcji postaci (IV) wykonaj:

AUTOMAT (ADR1,ADR2) . PRZEJSCIE ← ADR3
AUTOMAT (ADR1,ADR2) . WYJSCIE ← NRPROD
AUTOMAT (ADR1,ADR2) . REDUKCJA ← 1

gdzie ADR1, ADR3 są adresami odpowiednio "B" i "A" w TABSTAN, a ADR2 - adresem "C" w TABTERM.

Przykład 4.4. Podamy tablice TABTERM, TABSTAN oraz AUTOMAT skonstruowane przez procedurę LKONSTRUKTOR dla gramatyki z przykładu 4.1.

1	TERM1
2	a
3	b
4	d
5	e
6	g
7	i
8	n
9	0
10	1
11	letter
12	digit
13	:=

Tablica TABTERM

1	STAN 1
2	letter
3	digit
4	identifier
5	integer
6	leftside

Tablica TABSTAN

pole PRZEJSCIE

pole WYJSCIE

pole REDUKCJA

TERMINAL / STAN	(TERM1) 1	(a) 2	(b) 3	(d) 4	(e) 5	(g) 6	(i) 7	(n) 8	(o) 9	(1) 10	(letter 11	digit 12	(:= 13
(STAN1) 1		2^{14}_{0}	2^{15}_{0}	2^{16}_{0}	2^{17}_{0}	2^{18}_{0}	2^{19}_{0}	2^{20}_{0}	3^{21}_{0}	3^{22}_{0}			
(letter) 2	4^{6}_{0}												
(digit) 3	5^{11}_{0}												
(identifier) 4											4^{7}_{1}	4^{7}_{1}	6^{5}_{1}
(integer) 5												5^{12}_{1}	
(leftside) 6													

TABLICA AUTOMAT

4.4. Podmoduł L-analizatora

Podmoduł ten wykonuje następujące zadania:

a) czyta tekst programu źródłowego (tekst źródłowy);
b) wydziela w tekście źródłowym atomy leksykalne;
c) omija komentarze;
d) tablicuje teksty;
e) omija spacje i ciągi spacji.

Realizuje to procedura LANALIZATOR, która zawiera około 400 instrukcji w języku PL/1.

4.4.1. Procedura LANALIZATOR

Procedura ta jest wywoływana przez S-analizator w celu podania nowego symbolu potrzebnego do analizy syntaktycznej. Danymi wejściowymi procedury są tablice TABSTAN, TABTERM i AUTOMAT - skonstruowane przez L-konstruktor, zmienne COMBEG, COMEND, STRINGDEL - zdefiniowane przez projektanta języka oraz tekst programu źródłowego w postaci ciągu znaków. Na podstawie tych tablic i zmiennych LANALIZATOR łączy pojedyncze znaki tekstu źródłowego w atomy leksykalne, usuwa komentarze*), tablicuje teksty**) i ciągi spacji zastępuje jedną spacją***).

Procedura LANALIZATOR korzysta z trójelementowej tablicy STOS (elementy tej tablicy oznaczać będziemy przez STOS1, STOS2, STOS3). Każdy element tablicy STOS składa się z czterech

---

*) Komentarz jest to ciąg znaków umieszczony między znakami COMBEG i COMEND definiowanymi przez projektanta języka.

**) Tekst jest to ciąg znaków ograniczony z obu stron znakiem STRINGDEL, również definiowanym przez projektanta.

***) W różnych językach programowania spacje są odmiennie traktowane. Na przykład w Algolu 60 spacje są ignorowane poza tekstami, w Fortranie są znaczące (do 6 kolumny), w Pascalu są separatorami, a w Snobolu 4 - operatorami.
Aktualna wersja L-analizatora traktuje spacje jako separatory, tzn. łączy ciągi spacji występujące w programie (poza tekstami) w jedną spację, która zostaje przesłana do analizatora syntaktycznego.
W kolejnej wersji zostanie wprowadzona specjalna funkcja, za pomocą której projektant języka będzie mógł definiować sposoby analizowania spacji.

pól: tablicy PRODUKCJE - o zmiennym rozmiarze, dwóch pól liczbowych STAN i TERMINAL oraz pola znakowego TKN. Pola STAN i TERMINAL określają odpowiednio aktualny stan automatu i aktualnie analizowany symbol, pole TKN - ciąg znaków wejściowych już przeanalizowanych, a tablica PRODUKCJE zawiera identyfikatory produkcji użytych w dotychczasowej analizie.

Wyjściem procedury LANALIZATOR jest trójpolowa struktura ZWROT o polach TOKEN, TOKENS i WSK. Pole liczbowe TOKEN określa adres w tablicy SYMBTAB aktualnie skonstruowanego atomu leksykalnego, pole TOKENS zawiera atom leksykalny w postaci ciągu znaków, a pole WSK identyfikatory produkcji użytych do konstrukcji danego atomu (a właściwie adres tablicy zawierającej te identyfikatory).

4.4.2. Algorytm realizowany przez procedurę LANALIZATOR

Ponieważ program realizujący ten algorytm zawiera dużą liczbę zagnieżdżeń instrukcji warunkowych, jego opis w języku potocznym byłby zupełnie nieczytelny. Wobec tego podamy tylko bardzo ogólną ideę algorytmu, a jego działanie przedstawimy na przykładzie analizy programu źródłowego, napisanego w języku zdefiniowanym w przykładzie 4.1.

Procedura LANALIZATOR bada, czy w gramatyce istnieje produkcja, której lewa strona należy do V_R ($V_R \overset{df}{=} \{A \in V: \rho(A) \in I^o\}$), a prawa strona jest identyczna z pierwszymi dwoma (lub tylko z pierwszym) symbolami ciągu wejściowego. Jeżeli taka produkcja istnieje, to procedura LANALIZATOR wykonuje redukcję, tzn. symbole składające się na prawą stronę tej produkcji, zastępuje w ciągu wejściowym symbolem będącym lewą stroną tej produkcji.

Czynności te są powtarzane dopóty, dopóki nie można już dokonać żadnej redukcji; następuje wówczas przekazanie sterowania do programu realizującego moduł S-analizatora.

4.4.2.1. Przykład 4.5.

Analiza programu: begin␣ab1 := 11.01; end

Krok 1. Wczytujemy trzy pierwsze znaki programu. Otrzymujemy następującą sytuację:

	STOS1	STOS2	STOS3
PRODUKCJE	0	0	0
TERMINAL	3 (adres b w TABTERM)	5	6
STAN	1 (stan początkowy)	1	1
TKN	b	e	g

Krok 2. Ponieważ

AUTOMAT (1,3). PRZEJSCIE = 2
AUTOMAT (1,3). WYJSCIE = 15

wpisujemy

STOS.STAN ← 2
STOS1.PRODUKCJE ← 15
STOS1.TERMINAL ← 11 (jest to adres nazwy stanu-2 (czyli letter) w TABTERM).

Otrzymujemy następującą sytuację:

	STOS1	STOS2	STOS3
PRODUKCJE	14	0	0
TERMINAL	11	5	6
STAN	2	1	1
TKN	b	e	g

Krok 3. Ponieważ

STOS1.STAN ≠ 1

próbujemy analizować symbol zapisany w zmiennej STOS2.TERMINAL. Ale AUTOMAT (2,5). PRZEJSCIE = 0, próbujemy więc przejść do kolejnego stanu, niezależnie od analizowanego symbolu terminalnego. Ponieważ

AUTOMAT (2,1).PRZEJSCIE = 4
AUTOMAT(2,1).WYJSCIE = 6

wpisujemy

STOS1.STAN ← 4
STOS1.PRODUKCJE ← 6
STOS1.TERMINAL ← 0

otrzymujemy

	STOS1	STOS2	STOS3
PRODUKCJE	15,6	0	0
TERMINAL	0	5	6
STAN	4	1	1
TKN	b	e	g

Krok 4. Ponieważ

AUTOMAT (4,5). PRZEJSCIE = 0

oraz

AUTOMAT (4,1). PRZEJSCIE = 0

próbujemy wykonać krok pracy automatu "na STOSIE 2".

Krok 5. Ponieważ

AUTOMAT (1,5). PRZEJSCIE = 2
AUTOMAT (1,5). WYJSCIE = 17

otrzymujemy:

	STOS1	STOS2	STOS3
PRODUKCJE	15,6	17	0
TERMINAL	0	11	6
STAN	4	2	1
TKN	b	e	g

Krok 6. "Wracamy" do stanu zapamiętanego w zmiennej STOS1.STAN, analizując symbol zapisany w zmiennej STOS2.TERMINAL

AUTOMAT(4,11). PRZEJSCIE = 4
AUTOMAT(4,11). WYJSCIE = 7

otrzymujemy:

	STOS1	STOS2	STOS3
PRODUKCJE	15,6,7	17	0
TERMINAL	0	11	6
STAN	4	2	1
TKN	b	e	g

Ponieważ

AUTOMAT(4,11). REDUKCJA = 1

"dołączamy" zawartość zmiennej STOS2 do zawartości zmiennej STOS1, zawartość zmiennej STOS3 przepisujemy do zmiennej STOS2, a do zmiennej STOS3 przesyłamy następny znak wejściowy. Otrzymujemy:

	STOS1	STOS2	STOS3
PRODUKCJE	15,6,17,7	0	0
TERMINAL	0	6	7
STAN	4	1	1
TKN	be	g	i

Postępując analogicznie jak powyżej, otrzymamy w pewnej chwili następującą sytuację:

	STOS1	STOS2	STOS3
PRODUKCJE	15,6,17,7,18,7,19,7,20,7	0	0
TERMINAL	0	0	2
STAN	4	1	1
TKN	begin	␣	a

Ponieważ spacja jest traktowana jako ogranicznik, analiza ciągu wejściowego zostaje przerwana. Następuje sprawdzenie, czy atom zapisany w zmiennej STOS1.TKN jest słowem kluczowym. Następnie wypełniana jest struktura ZWROT następująco:

ZWROT.TOKEN ←—— 2 (adres "begin" w SYMBTAB)
ZWROT.WSK ——— adres pola STOS1.PRODUKCJE
ZWROT.TOKENS ←—— begin

po czym następuje powrót do S-analizatora.

Przy następnym wywołaniu L-analizatora, zostanie przesłana na wyjście spacja i otrzymamy następującą sytuację:

	STOS1	STOS2	STOS3
PRODUKCJE	0	0	0
TERMINAL	2	3	10
STAN	1	1	1
TKN	a	b	1

Kolejne wywołanie L-analizatora spowoduje następującą sytuację:

	STOS1	STOS2	STOS3
PRODUKCJE	14,6,15,7,22,8,5	22,11,22,12	0
TERMINAL	0	0	0
STAN	6	5	1
TKN	ab1:=	11	.

Ponieważ w tej sytuacji nie możemy wykonać kroku automatu na żadnym ze "STOSÓW", L-analizator kończy pracę, przesyłając na wyjście zawartość zmiennej STOS1, czyli

ZWROT.TOKEN = 7 (adres leftside w SYMBTAB)
ZWROT.WSK = adres pola STOS1.PRODUKCJE
ZWROT.TOKENS = ab1:=

Przy kolejnych wywołaniach L-analizatora otrzymamy następujące wyniki:

Wywołanie 4:

ZWROT.TOKEN = 13
ZWROT.WSK = 22,11,22,12 (są to oczywiście numery produkcji)
ZWROT.TOKENS = 11

Wywołanie 5:

ZWROT.TOKEN = 15
ZWROT.WSK = 0
ZWROT.TOKENS = .

Wywołanie 6:

ZWROT.TOKEN = 13
ZWROT.WSK = 21,11,22,12
ZWROT.TOKENS = 01

Wywołanie 7:

ZWROT.TOKEN = 6
ZWROT.WSK = 0
ZWROT.TOKENS = ;

Wywołanie 8:

ZWROT.TOKEN = 4
ZWROT.WSK = 0
ZWROT.TOKENS = end

Wywołanie 9:

ZWROT.TOKEN = 0
ZWROT.WSK = 0
ZWROT.TOKENS = $\perp$ (umowny znacznik końca programu).

4.4.3. Ograniczenia klasy gramatyk wejściowych

Wykażemy teraz, że dla pewnych klas gramatyk analiza leksykalna wykonywana według opisanego powyżej algorytmu*) nie będzie powodowała niejednoznaczności analizy syntaktycznej ciągu wejściowego.

*) W punkcie tym procedurę LANALIZATOR, realizującą powyższy algorytm, będziemy nazywać L-analizatorem (analizatorem leksykalnym).

Niech $G = \langle V_N, V_T, P, S \rangle$ będzie gramatyką bezkontekstową. Oznaczmy $V = V_N \cup V_T$, $V_R = \{A \in V: \varrho(A) \in I^0\}$.

Definicja 4.3

Niech $G = \langle V_N, V_T, P, S \rangle$

Zapis $\alpha \underset{*}{\Longrightarrow} \beta$ (dla $\alpha, \beta \in V^*$) oznacza, że istnieje taki ciąg $\gamma_1, \ldots, \gamma_n$ $(n \geqslant 1)$, że $\alpha = \gamma_1 \Longrightarrow \gamma_2 \Longrightarrow \gamma_3 \Longrightarrow \cdots \Longrightarrow \gamma_n = \beta$

Zapis $\alpha \Longrightarrow \beta$ (dla α, $\beta \in V^*$) oznacza, że $\alpha = \alpha_1 \varphi \alpha_2$, $\beta = \alpha_1 \psi \alpha_2$ i $(\varphi, \psi) \in P$

Słowo $\alpha \in V^*$ jest formą zdaniową gramatyki $G \Longleftrightarrow S \underset{*}{\Longrightarrow} \alpha$

Definicja 4.4

Gramatyka $G = \langle V_N, V_T, P, S \rangle$. Zdefiniujemy następujące relacje (tzw. relacje pierwszeństwa) pomiędzy symbolami S_1 i S_2 ze zbioru V:

1. $S_1 \doteq S_2$ jeżeli do P należy produkcja postaci: $U \rightarrow \alpha_1 S_1 S_2 \alpha_2$
2. $S_1 \lessdot S_2$ jeżeli
   (a) do P należy produkcja postaci $U \rightarrow \alpha_1 U_1 S_2 \alpha_2$ i przy tym $U_1 \underset{*}{\Longrightarrow} \alpha_3 S_1$
   lub (b) do P należy produkcja postaci $U \rightarrow \alpha_1 U_1 U_2 \alpha_2$ i przy tym $U_1 \underset{*}{\Longrightarrow} \alpha_3 S_1$ oraz $U_2 \underset{*}{\Longrightarrow} S_2 \alpha_4$
3. $S_1 \gtrdot S_2$ jeżeli do P należy produkcja postaci: $U \rightarrow \alpha_1 S_1 U_1 \alpha_2$ i $U_1 \underset{*}{\Longrightarrow} S_2 \alpha_3$

Gramatyka G jest gramatyką z pierwszeństwem jeżeli:

1. Dla każdej pary symboli S_1, S_2 z V zachodzi najwyżej jedna relacja pierwszeństwa.
2. Wszystkie produkcje ze zbioru P mają różne prawe strony.

Definicja 4.5

Niech $\alpha = \alpha_1 \alpha_2 \alpha_3 \cdots \alpha_n$ będzie formą zdaniową gramatyki G. Słowo $\alpha_i \alpha_{i+1} \cdots \alpha_{K-1} \alpha_K$ $(1 \leqslant i, K \leqslant n)$ jest rdzeniem formy α jeżeli:

(a) $\alpha_j \doteq \alpha_{j+1}$ $(i \leqslant j < K)$

(b) $\alpha_{i-1} \lessdot \alpha_i$

(c) $\alpha_K \gtrdot \alpha_{K+1}$

(d) nie istnieje ciąg $\alpha_p \alpha_{p+1} \cdots \alpha_{r-1} \alpha_r$ $(1 \leqslant p, r < i)$ taki, że

$\alpha_S \doteq \alpha_{S+1}$ $(p \leqslant S < r)$

$\alpha_{p-1} \lessdot \alpha_p$

$\alpha_r \gtrdot \alpha_{r+1}$

Ponieważ analiza syntaktyczna języka definiowanego przez gramatykę z pierwszeństwem polega na redukcji rdzeni kolejnych form zdaniowych, więc analiza leksykalna nie będzie powodowała niejednoznaczności, jeżeli również będzie redukowała tylko rdzenie.

Twierdzenie 4.3. Jeżeli gramatyka bezkontekstowa $G = \langle V_N, V_T, P, S \rangle$ spełnia następujące warunki:

(*) G jest gramatyką z pierwszeństwem,

(**) dla każdych dwóch produkcji p_1, $p_2 \in P$ takich, że $p_1 = (D, \gamma)$; $p_2 = (E, \beta_1 \gamma \beta_2)$; $\beta_1, \beta_2, \gamma \in V^*$; jeżeli $D \in V_R$ to $E \in V_R$

to analizator leksykalny zredukuje rdzeń ciągu wejściowego.

Dowód

Niech $\alpha = BC\alpha_1$ będzie aktualnie analizowaną przez L-analizator formą zdaniową, gdzie B, C, $\alpha_1 \in V^*$. Nie zmniejszając ogólności, możemy założyć, że B jest częścią rdzenia formy α. Ponieważ L-analizator redukuje najwyżej dwa symbole formy α, otrzymamy dwa przypadki, które rozważymy kolejno:

(a) L-analizator zredukuje w formie α symbol BC.

Wobec tego musi należeć do P produkcja: $A \longrightarrow BC$, gdzie $A, B, C \in V_R$. Pokażemy, że rdzeniem formy α jest rzeczywiście BC. Przypuśćmy, że tak nie jest. Rdzeniem formy α mogą być zatem: (1) $BC\alpha_1$ lub (2) B. W przypadku (1) musi należeć do P produkcja: $F \longrightarrow BC\,\alpha_1$. Oczywiście $F \notin V_R$ (ponieważ po prawej stronie produkcji występuje więcej niż dwa symbole), a więc otrzymujemy sprzeczność z założeniem (**) ($A \longrightarrow BC$, $F \longrightarrow BC\,\alpha_1$ należą do P, przy czym $A \in V_R$ i $F \notin V_R$). W przypadku (2) otrzymujemy natychmiast, że $B \circledcirc C$. Ponieważ jednocześnie $B \circledS C$ (ponieważ produkcja $A \longrightarrow BC$ należy do P), więc otrzymujemy sprzeczność z założeniem (*).

(b) L-analizator zredukuje w formie α symbol B.

Wobec tego produkcja $A \longrightarrow B$ należy do P; $A, B \in V_R$. Pokażemy, że rdzeniem formy α jest B. Przypuśćmy, że to nie zachodzi. Rdzeniem formy α mogą być zatem (1) $BC\alpha_1$ lub (2) BC. Przypadek (1) jest analogiczny jak (1) w punkcie (a). W przypadku (2) do P musi należeć produkcja $F \longrightarrow BC$. F nie może należeć do V_R, bo L-analizator konstruuje najdłuższy możliwy atom leksykalny i wobec tego zredukowałby słowa BC na F, a nie B na A. Wobec tego $F \notin V_R$ i otrzymujemy sprzeczność z założeniem (**).

cbdo.

Definicja 4.6

Określmy funkcję $h: V_R \longrightarrow I^0$ następująco:

1. $h(A) = 0 \Longleftrightarrow A \in V_T$ lub dla każdej produkcji $p = (A,\alpha) \in P$ p jest jednej z następujących postaci:
   $A \longrightarrow a$ lub $A \longrightarrow Ba$ lub $A \longrightarrow B$, gdzie
   $a \in V_T$, $B \in V_N$.

2. $h(A) = n \Longleftrightarrow$ n jest najmniejszą z możliwych liczb K takich, że dla każdej produkcji postaci $A \longrightarrow BC$ (gdzie $B \in V_N$, $C \in V_N$), $h(C) < K$.

Funkcję h nazywamy <u>głębokością regularności</u> gramatyki G.

Definicja 4.7

Gramatyka $G = \langle V_N, V_T, P, S \rangle$ jest gramatyką o głębokości regularności n $\Longleftrightarrow$ n jest najmniejszą z takich liczb K, że dla każdego $A \in V_R$ zachodzi $h(A) \leqslant K$.

Z przyjętej organizacji procedury LANALIZATOR wynika dodatkowe ograniczenie, aby głębokość regularności gramatyki wejściowej była nie większa niż 2 (analiza odbywa się w trójelementowej zmiennej STOS, w związku z czym są dostępne tylko trzy kolejne symbole analizowanej formy zdaniowej). Ograniczenie to przyjęto wyłącznie ze względów technicznych, nie zaś koncepcyjnych.

Praktycznie warunek ten nie ma specjalnego znaczenia, ponieważ większość gramatyk języków programowania spełnia to ograniczenie.

5. ZAKOŃCZENIE

Proponowana metoda pozwala uniknąć wielu wad występujących w stosowanych dotychczas rozwiązaniach, powoduje jednakże pewne zawężenie klasy języków wejściowych. Wydaje się jednak, że przy nieznacznej modyfikacji koncepcji ogólnej L-analizatora można by wykonywać analizę leksykalną dla dowolnego języka bezkontekstowego. Zmiana ta powinna polegać na tym, że S-analizator wywołując L-analizator wskazywałby mu jednocześnie, jaki atom leksykalny powinien pojawić się na wejściu. Pozwoliłoby to uniknąć niejednoznaczności napotykanych podczas analizy leksykalnej, które w opisanej realizacji są usuwane przez ograniczenia wymienione w punkcie 4.4.3.

Taka metoda byłaby optymalna, ponieważ L-analizator byłby generowany całkowicie automatycznie, a jednocześnie można by go było stosować do analizy wszystkich języków bezkontekstowych. Wymagałoby to jednak zmiany koncepcji również S-analizatora.

Chciałbym podziękować prof. dr hab. W.M. Turskiemu za liczne konsultacje i uwagi dotyczące pracy. Dziękuję również doc. dr hab. A.W. Mazurkiewiczowi, którego cenne uwagi pomogły poprawić pierwotną wersję pracy. Ponadto dziękuję p.p. J. Koncewicz-Krzemień, J. Borowcowi, A. Łukasiewiczowi i J. Witaszkowi za pomoc przy rozwiązywaniu wielu problemów związanych z pracą.

Bibliografia

[1] BLIKLE A.: Automaty i gramatyki. Wstęp do lingwistyki matematycznej, PWN, Warszawa 1971.

[2] COCKE J., SCHWARTZ J.T.: Programmi & Languages and Their Compilers. Preliminary notes. Courant Institute of Mathematical Sciences: New York University, 1970.

[3] FELDMAN J., GRIES D.: Translator Writing System, CACM, 1968, 2, 77-113.

[4] GRIES D.: Compiler Construction for Digital Computers, John Wiley and Sons, Inc. 1971.

[5] HOPCROFT J.: An NlogN Algorithm for Minimising States in a Finite Automaton, STAN-CS-71-190. 1971.

[6] HOPCROFT J., ULLMAN J.D.: Formal Languages and Their Relation to Automata, Addison-Wesley Publishing Company, Inc. 1969.

[7] HOPGOOD F.R.A.: Compiling Techniques, American Elsevier Publishing Company, Inc. New York 1969.

[8] HOPGOOD F.R.A., BELL A.G.: The ATLAS ALGOL Preprocessor for non Standard Dialects. 1967 CJ, 4, 360-364.

[9] IGLEWSKI M., KRZEMIEŃ R.: Analiza leksykalna, ETO Nowości 1974, 3.

[10] INGERMAN P.Z.: A Syntax-Oriented Translator, Academic Press, 1966.

[11] JOHNSON W.L., PORTER J.H., ACKLEY S.I., ROSS D.T.: Automatic Generation of Efficient Lexical Processors Using Finite State Techniques, CACM, 1968, 12, 805-813.

[12] JOHANSEN P.: Construction of Recognition Devices for Regular Languages from Their Backus Normal Form Definition, BIT, 1966, 6, 294-309.

[13] KNUTH D.E.: Sorting and Searching: the Art of Computer Programming, Addison-Wesley Publishing Co., Inc. 1970, t. 3.

[14] KONCEWICZ J.: Preliminary Analysis of ALGOL-60 Source Programs for ZAM Computers, Algorytmy, 1969, 10, 77-88.

[15] LECARME O.: Un générateur d'analyseurs lexicaux. Doc. de travail: Université de Montreal, 1973, nr 40.

[16] LYNCH W.C., PIERSON H.L.: A Finite State Transducer Model for Compiler Lexical Scanners, IFIP 68, North-Holland Publishing Co., Amsterdam 1969.

[17] MALUSZYŃSKI J.: Podgramatyki elementarne składni Algolu 60, Algorytmy, 1970, 13, 17-34.

[18] Mc CLURE R.M.: TMS - a Syntax Directed Compiler, Proceedings ACM National Conf., 1965.

[19] NAUR P. ed.: Revised Report on the Algorithmic Language ALGOL-60, CACM, 1962, 1, 1-17.

[20] PLASKOV J., SCHUMAN S.: The TRANGEN System on the M460 Computer, Computer Associates Inc. Report, 1966.

[21] REYNOLDS J.C.: COGENT Programming Manual, ANL-7022 Report, 1965.

[22] ROSEN S.: A Compiler Building System Developed by Brooker and Morris, CACM, 1964, 7, 403-414.

[23] ROSS D.T.: The AED Approach to Generalized Computer-Aided Design, Report ESL-R-305, MIT, 1967.

[24] SCHEIDIG H.: ATWS: A Translator Writing System, Bericht nr 7219, TUM, 1971.

[25] SQUIRES B.E.: Lexical Analysis by a Precedence Grammar, University Illinois Dept. Computer Science Report, 1966.

[26] TURSKI W.M.: Struktury danych, WNT, Warszawa 1971.

[27] WALIGÓRSKI S.: Algebraiczna teoria automatów, Algorytmy, 1969, 11.

[28] WIRTH N., WEBER H.: EULER - a Generalization of ALGOL, and its Formal Definition, CACM, 1966, 1, 13-25: CACM, 1966, 2, 89-99.

[29] Zakład Teorii Translatorów IMM: Projekt koncepcyjny metatranslatora, Raport wewnętrzny, 1974.

DODATEK A. Program realizujący moduł analizy leksykalnej

```
    RCMAN: PROCEDURE OPTIONS(MAIN);
NEST
            RCMAN: PROCEDURE OPTIONS(MAIN);
                   DCL M1 FIXED BINARY;
                   DCL M2 FIXED BINARY;
                   DCL M5 FIXED BINARY;
                   DCL N  FIXED BINARY;
                   DCL I  FIXED BINARY;
                   DCL ILST FIXED BINARY;
                   DCL ILT  FIXED BINARY;
                   DCL A(1:80) CHAR(1);
                   DCL 1 SYMTAB(1:N) CONTROLLED,
                       2 SYMBOL CHAR(20) VARYING,
                       2 TCNT BIT(1);
                   DCL PROCAT(1:N) POINTER CONTROLLED;
                   DCL FCNR(1:N) BIT(1) CONTROLLED;
                   DCL STARTSYMB FIXED BINARY;
                   DCL 1 PROC BASED(P),
                       2 ILOSC FIXED BINARY,
                       2 ADRALT(1:M) REFER(ILOSC)) POINTER;
                   DCL 1 ALTER BASED(R),
                       2 CALT  FIXED BINARY,
                       2 ICALT FIXED BINARY,
                       2 SYMBOL(1:M2 REFER(CALT)) FIXED BINARY;
                   DCL SYSIN RECORD INPUT;
                   DCL TABTERM(1:ILT) CHAR(20) VARYING CONTROLLED;
                   DCL TABSTAN(1:ILST) CHAR(20) VARYING CONTROLLED;
                   DCL 1 AUTOMAT(1:ILST,1:ILT) CONTROLLED,
                       2 PRZEJSCIE FIXED BINARY,
                       2 WYJSCIE FIXED BINARY, /* NR PRODUKCJI*/
                       2 REDUKCJA BIT(1);
                   DCL 1 ZWROT,  /*DLA  ANALIZATORA SYNT */
                       2 TOKEN FIXED BINARY,
                       2 WSK POINTER,
                       2 TOKENS CHAR(20)VARYING;
                   DCL Q POINTER;
                   DCL 1 STOS(1:2),
                       2 WSKAZNIK POINTER, /*POINTER DO PRODUKCJI*/
                       2 TERMINAL FIXED BINARY,
                       2 STAN FIXED BINARY,
                       2 TKN  CHAR(20) VARYING;
                   DCL 1 PRODUKCJE BASED(P),
                       2 LRPROD FIXED BINARY,
                       2 ARPROD(1:M5 REFER(LRPROD)) FIXED BINARY;
                   M1=1;
                   M2=1;
                   M5=20;
                   I =0;
                            N=160;
                            ILT=45;
                            ILST=6;
                   ALLOCATE PROCAT;
                   ALLOCATE SYMTAB;
                   ALLOCATE AUTOMAT;
                   ALLOCATE TABSTAN;
                   ALLOCATE TABTERM;
                   ALLOCATE RCNR;
                   DO J=1 TO N;
```

```
RCHAR: PROCEDURE OPTIONS(MAIN);

ACST

  1                    PROCAT(J)=NULL;
  1                    SYMETAB.SYMBOL(J)='';
  1                    SYMETAB.TCNT(J)='1'B;
  1                    RCHR(J)='0'B;
  1                 END;
                    SYMETAB.SYMBOL(1)='SPACJA';
                    OPEN FILE(SYSIN);
                    ON ENDFILE(SYSIN) GO TO KONIEC;
                    CALL LACCHACZ;
                    PUT DATA (SYMETAB);
                    RCHR=SYMETAB.TCNT;
                    PUT DATA (RCHR);
                    CALL REGULAR;
                    CALL LKONSTRUKTOR;
                    PUT DATA (SYMETAB);
                    PUT DATA (STARTSYMB);
                PUT DATA (RCHR);
                    PUT DATA (ILT);
                    PUT DATA (TABTERM);
                    PUT DATA (ILST);
                    PUT DATA (TABSTAN);
                    PUT DATA (AUTOMAT);
                CALL  LANALINIT;
                PUT DATA(ZWROT.TOKENS,ZWROT.TOKEN);
                  PU1: CALL LANALIZATOR;
                PUT   DATA(ZWROT.TOKENS,ZWROT.TOKEN);
                    GO TO PU1;
            LACCHACZ: PROCEDURE;
                    DCL ARRROW FIXED BINARY;
                    DCL 1 WYNIK,
                        2 NAZWA CHAR(20) VARYING,
                        2 TNT BIT(1);
                    DCL ADRES  FIXED BINARY;
                    DCL PROC1(1:30) POINTER;
                    DCL LFS FIXED BINARY;
                    DCL RFS(1:30) FIXED BINARY;
                    PUT LIST ('LACCHACZ');
                    READ FILE(SYSIN) INTO(A);
                    PUT DATA (A);
                  L1: I=I+1;
                    CALL NUMER;
                    CALL TOKEN;
                    IF WYNIK.TNT='1'B THEN GO TO BLAD1;
                    CALL SPACJA;
                    IF A(I)¬='=' THEN GO TO BLAD2;
                    I=I+1;
                    CALL PASF;
                    LFS=ADRES;
                  L2: CALL TOKEN;
                    CALL PASF;
                    RFS(N2)=ADRES;
                       RCHR(ADRES)='1'B;
                    CALL SPACJA;
                    IF A(I)=',' THEN
                         GO;
```

```
   ROMAN: PROCEDURE OPTIONS(MAIN);

NEST

  1                    PUT LIST('PRZECINEK');
  1                    M2=M2+1;
  1                    I=I+1;
  1                    GO TO L2;
  1                    END;
                   IF A(I)=';' THEN
                        DO;
  1                    CALL SLASH;
  1                    CALL SREDNIK;
  1                    GO TO L1;
  1                    END;
                   IF A(I)='|' THEN
                        DO;
  1                    CALL SLASH;
  1                    I=I+1;
  1                    CALL NUMER;
  1                    M1=M1+1;
  1                    GO TO L3;
  1                    END;
                   IF A(I)='.' THEN
                        DO;
  1                    PUT LIST('KROPKA');
  1                    CALL SLASH;
  1                    CALL SREDNIK;
  1                    CALL AKSJOMAT;
  1                    RETURN;
  1                    END;
          AKSJOMAT: PROCEDURE;
              DO J=1 TO N;
  1                     IF(RDNR(J)='0'B & TCNT(J)='0'B) THEN
  1                    DO;
  2                    STARTSYMB=J;
  2                    RETURN;
  2                    END;
  1                 END;
                    GO TO BLAD6;
                    END AKSJOMAT;
                    PUT LIST ('SREDNIK');
           SREDNIK: PROCEDURE;
                    ALLOCATE PROD;
                    PROCAT(LHS)=P;
                    DO J=1 TO M1;
  1                   PROD.ACPALT(J)=PROD1(J);
  1                 END;
                    M1=1;
                    END SREDNIK;
             SLASH: PROCEDURE;
                    PUT LIST ('SLASH');
                    ALLOCATE ALTER;
                    PROD1(M1)=P;
                    ALTER.ICALT=ARPROC;
                    DO J=1 TO M2;
  1                    ALTER.SYMBOL(J)=RHS(J);
  1                 END;
                    M2=1;
```

```
PCMAN: PROCEDURE OPTIONS(MAIN);

TEST

                    END SLASH;
               HASH: PROCEDURE;
                    DCL H1 FIXED BINARY;
                    DCL H2 CHAR(20) VARYING;
                    DCL H3 CHAR(20) VARYING;
                    PUT LIST ('HASH');
                    H1=1;
                    H2=WYNIK.NAZWA;
                 H4: H3=SYMETAB.SYMBOL(H1);
                    IF H2=H3 THEN GO TO H5;
                    IF H3=' ' THEN
                         DO;
1                        SYMETAB(H1)=WYNIK;
1                        GO TO H5;
1                        END;
                    IF H1<N THEN
                         DO;
1                        H1=H1+1;
1                        GO TO H4;
1                        END;
                            ELSE GO TO BLAD3;
                 H5: ADRES=H1;
                    END HASH;
            SPACJA: PROCEDURE;
                    PUT LIST ('SPACJA');
                 S1: IF A(I)=' ' THEN
                         DO;
1                        IF I<80 THEN
1                             DO;
2                            I=I+1;
2                            GO TO S1;
2                            END;
1                               ELSE
1                            DO;
2                            READ FILE(SYSIN) INTO(A);
2                            PUT DATA(A);
2                            I=1;
2                            GO TO S1;
2                            END;
1                    END SPACJA;
            NUMER: PROCEDURE;
                    DCL CYFRA CHAR(10) INIT('1234567890');
                    DCL N1 FIXED BINARY;
                    DCL N2 FIXED BINARY;
                    PUT LIST ('NUMER');
                    N1=0;
                    CALL SPACJA;
                 N3: N2=INDEX(CYFRA,A(I));
                    IF N2^=0 THEN
                         DO;
1                         IF N2=10 THEN N2=0;
1                        N1=N1*10+N2;
1                        I=I+1;
1                        GO TO N3;
1                        END;
```

```
   PCHAN: PROCEDURE OPTIONS(MAIN);

NEST

                        ELSE
                     DO;
1                     CALL SPACJA;
1                     IF A(I)¬=':' THEN GO TO BLAD4;
1                                  ELSE
1                        DO;
2                        NRPROC=N1;
2                        I=I+1;
2                        PUT DATA(NRPROC);
2                  END NUMER;
            TOKEN: PROCEDURE;
                  DCL LITERA CHAR(26) INIT('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
           DCL IDENT CHAR(37) INIT('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-');
                  DCL 1 T1,
                      2 T2 CHAR(20) VARYING,
                      2 T3 BIT(1);
                  PUT LIST ('TOKEN');
                  T1.T2='';
                  CALL SPACJA;
                  IF A(I)='''' THEN
               T5:      BEGIN;
                        I=I+1;
                        IF A(I)¬='''' THEN
                           DO;
1                           T1.T2=T1.T2||A(I);
1                           GO TO T5;
1                          END;
                                    ELSE
                           DO;
1                           T1.T3='1'B;
1                           WYNIK=T1;
1                           I=I+1;
1                           RETURN;
1                          END;
                     END;
                        ELSE
                     DO;
1                     IF VERIFY (A(I),LITERA)¬=0 THEN GO TO BLAD5;
1                     T1.T2=T1.T2||A(I);
1              T6:    I=I+1;
1                     IF VERIFY (A(I),IDENT)=0 THEN
1                        DO;
2                        T1.T2=T1.T2||A(I);
2                        GO TO T6;
2                       END;
2                                                ELSE
1                        DO;
2                        T1.T3='0'B;
2                        WYNIK=T1;
2               T7:  END TOKEN;
          LKONIEC:  END LACZNAC2;
          REGULAR:  PROCEDURE;
                  DCL ILOSCC FIXED BINARY;
                  DCL BLOKSK BIT(1);
                  DCL BLOALT BIT(1);
```

```
REPAR: PROCEDURE OPTIONS(MAIN);

%EST

                    REGWSK='0'B;
                R7: DO J=1 TO N;
1                   IF RCNR(J) ='0'B THEN
1                   DO;
2                   P=PRDCAT(J);
2                   ILPROC=PROC.ILOSC;
2                   CALL REG1;
2                   RCNR(J)=REGALT;
2                   REGWSK =(REGALT|REGWSK);
2                       END;
1                   END;
                    IF REGWSK='1'B THEN
                        DO;
1                       PUT DATA(RCNR);
1                       REGWSK='0'B;
1                       GO TO R7;
1                       END;
                        REGWSK='1'B;
                R8: DO J=1 TO N;
1                   IF(RCNR(J)='1'B & TCNT(J)='0'B) THEN
1                       DO;
2                       P=PRDCAT(J);
2                       ILPROC =PROC.ILOSC;
2                       CALL REG2;
2                       RCNR(J)=REGALT;
2                       REGWSK=(REGALT & REGWSK);
2                       END;
1                   END;
                    IF REGWSK='0'B THEN
                        DO;
1                       PUT DATA(RCNR);
1                       REGWSK='1'B;
1                       GO TO R8;
1                       END;
            REG1: PROCEDURE;
                  DO K=1 TO ILPROC;
1                   R=PROC.ADRALT(K);
1                   CALL REG2;
1                   IF REGALT='0'B THEN RETURN;
1                 END;
                  END REG1;
            REG2: PROCEDURE;
                  DCL PCE1 FIXED BINARY;
                  DCL PCE2 FIXED BINARY;
                  REGALT='0'B;
                  IF ALTER.CALT>2 THEN RETURN;
                  IF ALTER.CALT=1 THEN
                  DO;
1                   PCE1=ALTER.SYMPCL(1);
1                 REGALT=RCNR(PCE1);
1                 RETURN;
1                 END;
                                    ELSE
                  PCE2=ALTER.SYMPCL(2);
                  PCE1=ALTER.SYMPCL(1);
```

```
RCHAR: PROCEDURE OPTIONS(MAIN);

NFST

                    REGALT=(¬TCNT(RCB1)& RCHR(RCB2));
                    END REG2;
               REG3: PROCEDURE;
                    DO K=1 TO ILPROD;
  1                    R=PROD.ADRALT(K);
  1                    CALL REG4;
  1                 IF REGALT='0'B THEN RETURN;
  1                 END;
                    END REG3;
               REG4: PROCEDURE;
                    DCL RCB1 FIXED BINARY;
                    RCB1=ALTER.SYMBOL(1);
                    REGALT=RCHR(RCB1);
                    END REG4;
                    END REGULAR;
          LKONSTRUKTOR:PROCEDURE;
                    DCL LHS FIXED BINARY;
                    DCL RHS1 FIXED BINARY;
                    DCL RHS2 FIXED BINARY;
                    DCL ILPRC FIXED BINARY;
                    DCL LK2 CHAR(20) VARYING;
                    DCL NRPROD FIXED BINARY;
                    DCL LK3 FIXED BINARY;
                    PUT LIST ('LKONSTRUKTOR');
                    DO J=1 TO ILT;
  1                  TABTERM(J)='';
  1                 END;
                    DO J=1 TO ILST;
  1                   TABSTAN(J)='';
  1                  DO K=1 TO ILT;
  2                   AUTOMAT(J,K).PRZEJSCIE=0;
  2                   END;
  1                 END;
                    TABTERM(1)='TERM1';
                    TABSTAN(1)='STAN1';
                    DO J=1 TO N;
  1                   IF (RCHR(J)='1'B & TCNT(J)='0'B) THEN
  1                      DO;
  2                      P=PRODAT(J);
  2                      ILPRC=PROD.ILOSC;
  2                      LK2=SYMBTAB.SYMBOL(J);
  2                      CALL SEARCH(TABSTAN,LHS,ILST);
  2                           DO K=1 TO ILPRC;
  3                           P=PROD.ADRALT(K);
  3                           NRPROD=ALTER.IDALT;
  3                           LK3 =ALTER.SYMBOL(1);
  3                           LK2 =SYMBTAB.SYMBOL(LK3);
  3                           IF ALTER.CALT=2 THEN
  3                              DO;
  4                              CALL SEARCH(TABSTAN,RHS1,ILST);
  4                              LK3=ALTER.SYMBOL(2);
  4                              LK2=SYMBTAB.SYMBOL(LK3);
  4                              CALL SEARCH(TABTERM,RHS2,ILT);
  4                              AUTOMAT(RHS1,RHS2).PRZEJSCIE=LHS;
  4                              AUTOMAT(RHS1,RHS2).WYJSCIE=NRPROD;
```

```
   RCHAN: PROCEDURE OPTIONS(MAIN);


NEST

  4                                  AUTOMAT(RHS1,RHS2).REDUKCJA='1'B;
  4                                  END;
  3                                           ELSE /* TZN.CAL1=1*/
  3                                  DO;
  4                              IF     SYMTAB.TCN1(LK2)='1'B THEN
  4                                     DO;
  5                                     CALL SEARCH(TABTERM,RHS1,ILT);
  5                                     AUTOMAT(1,RHS1).PRZEJSCIE=LHS;
  5                                     AUTOMAT(1,RHS1).WYJSCIE=NRPROD;
  5                                     AUTOMAT(1,RHS1).REDUKCJA='0'B;
  5                                     END;
  4                                                    ELSE
  4                                     DO;
  5                                     CALL SEARCH(TABSTAN,RHS1,ILST);
  5                                     AUTOMAT(RHS1,1).PRZEJSCIE=LHS;
  5                                     AUTOMAT(RHS1,1).WYJSCIE=NRPROD;
  5                                     AUTOMAT(RHS1,1).REDUKCJA='0'B;
  5                                     END;
  4                                  END;
  3                              END;
  2                         END;
  1                    END;
          SEARCH: PROCEDURE(TAB,ADR,WYM);
                  DCL TAB(*) CHAR(20) VARYING;
                  DCL ADR FIXED BINARY;
                  DCL WYM FIXED BINARY;
                  DCL PCB1 CHAR(20) VARYING;
                  PUT LIST('SEARCH');
                  DO I=1 TO WYM;
  1                PCB1=TAB(I);
  1                IF PCB1=LK2 THEN
  1                   DO;
  2                   ADR=I;
  2                   RETURN;
  2                   END;
  1                IF PCB1='' THEN
  1                   DO;
  2                   ADR=I;
  2                   TAB(I)=LK2;
  2                   RETURN;
  2                END;
  1                END;
                   END SEARCH;
                   END LKONSTRUKTOR;
          LANALINIT: PROCEDURE;
                   DCL B FIXED BINARY;
                   DCL REDUKUJ BIT(1);
                   DCL SZT CHAR(30) VARYING;
                   DCL LA4 FIXED BINARY;
                   DCL ZNPOTNICA BIT(1);
                   DCL LA6 FIXED BINARY;
                   /* PO DOLACZENIU DO LKONSTR DODAC DEKLARACJE  A,I*/
                   /* SYSIN, SYMTAB, ILT, ILST, TABTERM  ITD */
                   ON ENDFILE(SYSIN) DO TO KONIECLA;
                   READ FILE(SYSIN) INTO(A);
```

```
   ROMAN: PROCEDURE OPTIONS(MAIN);

NEST

                    I=1;
                    DO J=1 TO 2;
  1                 ALLOCATE PROCESY;
  1                 WSKAZNIK(J)=F;
  1                 LPROC=1;
  1                 NRPROC(LPROC)=0;
  1                 STOS.STAN(J)=1;
  1                 TKN(J)=A(I);
  1                   SZT=A(I);
  1                   CALL SZUKTERM;
  1                   TERMINAL(J)=LA4;
  1                 I=I+1;
  1                 END;
                    ALLOCATE PROCESY;
                    WSK=F;
        LEKALIZATOR: ENTRY;
               LA01: BEGIN;
                    DCL LA02 BIT(1);
                    LA02='0'B;
               LA0: DO WHILE (TKN(1)=' ');
  1                 LA02='1'B;
  1               SP1:      STOS(1)=STOS(2);
  1                         STOS(2)=STOS(3);
  1                         F=WSKAZNIK(2);
  1                         LPROC=1;
  1                         NRPROC(LPROC)=0;
  1                         STAN(2)=1;
  1                         IF I>80 THEN
  1                              DO;
  2                              READ FILE(SYSIN) INTO(A);
  2                              I=1;
  2                              END;
  1                         SZT=A(I);
  1                         CALL SZUKTERM;
  1                         TERMINAL(2)=LA4;
  1                         TKN(2)=A(I);
  1                         I=I+1;
  1                   END;
                    IF LA02='1'B THEN DO;
  1                      ZWROT.TOKEN=1;
  1                      ZWROT.WSK=NULL;
  1                      ZWROT.TOKENS=' ';
  1                      RETURN;
  1                        END;
                         END;
               LA1: B=1;
                    CALL ANAL;
                    IF ZWROTNICA='1'B THEN GO TO LA1;
               LA2: B=2;
                    CALL ANAL;
                    IF ZWROTNICA='1'B THEN GO TO LA1;
                    B=3;
                    CALL ANAL;
                    IF ZWROTNICA ='1'B THEN GO TO LA2;
                    IF (STOS.STAN(1) & STOS.STAN(2)=1) THEN
```

```
ROMAN: PROCEDURE OPTIONS(MAIN);

NEST

                        DO;
1                       TOKENS=TKN(1)||TKN(2);
1                       CALL KEYWORD;
1                       IF TOKEN ¬=0  THEN
1                         DO;
2                         P=WSK;
2                         LPROC=1;
2                         NPROC(LPROC)=0;
2                         STOS(1)=STOS(3);
2                           DO J=2 TO 3;
3                           P=WSKAZNIK(J);
3                           LPROC=1;
3                           NPROC(LPROC)=0;
3                           STAN(J)=1;
3                           IF I >80 THEN
3                             DO;
4                             READ FILE (SYSIN) INTO(A);
4                             I=1;
4                             END;
3                           TKN=A(I);
3                       SZT=A(1);
3                       CALL SZUKTERM;
3                       TERMINAL(J)=LA4;
3                             I=I+1;
3                             END;
2                         RETURN;
2                         END;
1                       END;
                      IF ¬(STOS.STAN(2)=1 & STOS.STAN(3)=1) THEN
                        DO;
1                       TOKENS=TKN(2)||TKN(3);
1                       CALL KEYWORD;
1                       IF TOKEN ¬=0 THEN
1                         DO;
2                         TKN(2)=TOKENS;
2                       SZT=TOKENS;
2                       CALL SZUKTERM;
2                       TERMINAL(2)=LA4;
2                         P=WSKAZNIK(2);
2                         LPROC=1;
2                         NPROC(LPROC)=0;
2                         IF I>80 THEN
2                           DO;
3                           READ FILE(SYSIN) INTO(A);
3                           I=1;
3                           END;
2                       TKN(3)=A(I);
2                       SZT=A(1);
2                       CALL SZUKTERM;
2                       TERMINAL(2)=LA4;
2                         I=I+1;
2                         P=WSKAZNIK(3);
2                       LPROC=1;
2                         NPROC(LPROC)=0;
2                         GO TO LA1;
```

```
   ROMAN: PROCEDURE OPTIONS(MAIN);

NEST

 2                      END;
 1                   END;
                  TOKENS=TKN(1);'
                  CALL KEYWORD;
         IF((TOKEN¬=0)&(SYMETAB.TCNT(TOKEN)='0'B)) THEN
                      DO;
 1                    F=WSK;
 1                    LPROC=1;
 1                    NPROC(LPROC)=0;
 1                    END;
                           ELSE
                      DO;
 1                    TOKENS=TABSTAN(STAN(1));
 1                    CALL KEYWORD;
 1                    TOKENS=TKN(1);
 1                    F=STOS.WSKAZNIK(1);
 1                    WSK->PROCEDURE =PROCEDURE;
 1                    LPROC=LPROC-1;
 1                    END;
                  STOS(1)=STOS(2);
                  STOS(2)=STOS(3);
                  F=WSKAZNIK(3);
                  LPROC=1;
                  NPROC(LPROC)=0;
                  STAN(3)=1;
                  IF I>80  THEN
                      DO;
 1                    READ FILE (SYSIN) INTO(A);
 1                    I=1;
 1                    END;
                    SZT=A(I);
                    CALL SZUKTERM;
                    TERMINAL(3)=LA4;
                  TKN(3)=A(I);
                  I=I+1;
                  RETURN;
            ANAL: PROCEDURE;
                  IF STOS.STAN(B)=1 THEN
                      DO;
 1                    IF STOS.TERMINAL(B)=0 THEN  /*CZY ZNAKOWY SYMBOL*/
 1                       DO;
 2                  IF((B=3)|(TERMINAL(B+1)¬=0)|(STOS.STAN(B+1)¬=1)) THEN
 2                             DO;
 3                             ZWROTNICA='0'B;
 3                             RETURN;
 3                             END;
 2                         TOKENS=TKN(B) ||TKN(B+1);
 2                         CALL KEYWORD;
 2                             IF TOKEN=0 THEN
 2                             DO;
 3                             ZWROTNICA='0'B;
 3                             RETURN;
 3                             END;
 2                         TKN(B)=TOKENS;
 2                    SZT=TOKENS;
```

```
SCAN: PROCEDURE OPTIONS(MAIN);

EST

2                    CALL SZUKTERM;
2                    TERMINAL(P)=LA4;
2                            IF P=1 THEN STOS(2)=STOS(2);
2                            IF I>80 THEN
                                 DO;
3                                READ FILE(SYSIN) INTO(A);
3                                I=1;
3                                END;
2                    SZT=A(I);
2                    CALL SZUKTERM;
2                    TERMINAL(2)=LA4;
2                            TKN(2)=A(I);
2                            STAN(2)=1;
2                            P=WSKAZNIK(2);
2                            LPROD=1;
2                            NPROD(LPROD)=0;
2                            ZWROTNICA='1'B;
2                            RETURN;
2                            END;
1                       IF AUTOMAT(1,TERMINAL(P)).PRZEJSCIE=0 THEN
1                            DO;
2                            ZWROTNICA='0'B;
2                            RETURN;
2                            END;
1                                                          ELSE
1                            DO;
2                       STOS.STAN(P)=AUTOMAT(1,TERMINAL(P)).PRZEJSCIE;
2                            P=WSKAZNIK(P);
2                                 DO L=2 TO LPROD;
3                                 NPROD(L)=NPROD(L-1);
3                                 END;
2                            NPROD(1)=AUTOMAT(1,TERMINAL(P)).WYJSCIE;
2                            LPROD=LPROD+1;
2                            TOKENS=TABSTAN(STOS.STAN(P));
2                    SZT=TOKENS;
2                    CALL SZUKTERM;
2                    TERMINAL(P)=LA4;
2                            ZWROTNICA='1'B;
2                            RETURN;
2                            END;
1                       END;
                                 ELSE /*TZN STAN ~=1 */
                            DO;
1                           IF P=3 THEN
1                                DO;
2                                ZWROTNICA='0'B;
2                                RETURN;
2                                END;
1                  IF AUTOMAT(STOS.STAN(P),TERMINAL(P+1)).PRZEJSCIE=0 THEN
1                  IF AUTOMAT(STOS.STAN(P),1).PRZEJSCIE=0  THEN
1                            DO;
2                            ZWROTNICA='0'B;
2                            RETURN;
2                            END;
1                                                          ELSE
```

```
ROMAN: PROCEDURE OPTIONS(MAIN);

ARST

 1•                        CC;
 2                  RECUKUJ=AUTOMAT(STAN(B),1).RECUKCJA;
 2                       STOS.STAN(B)=AUTOMAT(STOS.STAN(B),1).PRZEJSCIE;
 2                                    TOKENS=TABSTAN(STOS.STAN(B));
 2                    SZT=TOKENS;
 2                    CALL SZUKTERM;
 2                    TERMINAL(B)=LA4;
 2                         F=STOS.WSKAZNIK(B);
 2                              DO L=2 TO LPROC;
 3                              NRPROC(L)=NRPROC(L-1);
 3                              END;
 2                         NRPROC(1)=AUTOMAT(STOS.STAN(B),1).WYJSCIE;
 2                       IF RECUKUJ='C'B THEN
 2                              GO TO LA6;
 2                                   ELSE
 2                              GO TO LA5;
 2                         END;
 1                                                                  ELSE
 1                         CC;  /* ANALIZA STOS(1),STOS(2)*/
 2               RECUKUJ=AUTOMAT(STAN(B),TERMINAL(B+1)).RECUKCJA;
 2               STOS.STAN(B)=AUTOMAT(STOS.STAN(B),TERMINAL(B+1)).PRZEJSCIE;
 2                         TOKENS=TABSTAN(STOS.STAN(B));
 2                  SZT=TOKENS;
 2                  CALL SZUKTERM;
 2                  TERMINAL(B)=LA4;
 2                         F=STOS.WSKAZNIK(B);
 2                              DO L=2 TO LPROC;
 3                              NRPROC(L)=NRPROC(L-1);
 3                              END;
 2                    NRPROC(1)=AUTOMAT(STOS.STAN(B),TERMINAL(B+1)).WYJSCIE;
 2                    IF RECUKUJ='C'B THEN
 2                                             GO TO LA6;
 2                                        ELSE GO TO LA5;
 2                         END;
 1          LA9:      TKN(B)=TKN(B)||TKN(B+1);
 1                    IF B=1 THEN
 1                         CC;
 2                         F=WSKAZNIK(1);
 2                         C=WSKAZNIK(2);
 2                         LA6=C->LPROC-1;
 2                              DO K=1 TO LA6;
 3                              NRPROC(LPROC)=C->NRPROC(K);
 3                              LPROC=LPROC+1;
 3                              END;
 2                         STOS(2)=STOS(3);
 2                              C=WSKAZNIK(3);
 2                         END;
 1                           ELSE
 1                         CC;
 2                         F=WSKAZNIK(2);
 2                         C=WSKAZNIK(3);
 2                         LA6=C->LPROC-1;
 2                              DO K=1 TO LA6;
 3                              NRPROC(LPROC)=C->NRPROC(K);
 3                              LPROC=LPROC+1;
```

```
   PCMAN: PROCEDURE OPTIONS(MAIN);

NEST
  2                                   END;
  2                                   END;
  1                          C->LRPROD=1;
  1                          STAN(2)=1;
  1                          IF I>80 THEN
  1                              DO;
  2                              READ FILE(SYSIN) INTO(A);
  2                              I=1;
  2                              END;
  1                          P=WSKAZNIK(2);
  1                      TKN(2)=A(I);
  1                        SZT=A(I);
  1                        CALL SZUKTERM;
  1                        TERMINAL(2)=LA4;
  1                          I=I+1;
  1                          LRPROD=1;
  1                          NRPROD(LRPROD)=0;
  1                          GO TO LA10;
  1                LA8:      LRPROD=LRPROD+1;
  1               LA10:      ZWROCTNICA='1'B;
  1                          RETURN;
  1                          END;
                       END ANAL;
              KONIECLA: TOKEN=0;
                       P=WSK;
                       LRPROD=1;
                       NRPROD(LRPROD)=0;
                       RETURN;
              SZUKTERM: PROCEDURE;
                          DO M=1 TO ILT;
  1                       IF TABTERM(M)=SZT THEN DO;
  2                                                LA4=M;
  2                                                RETURN;
  2                                                END;
  1                          END;
                          LA4=0;
                       END SZUKTERM;
               KEYWORD: PROCEDURE;
                          DO M=1 TO N;
  1                       IF SYMTAB.SYMBOL(M)=TOKENS THEN
  1                             DO;
  2                             TOKEN=M;
  2                             RETURN;
  2                             END;
  1                          END;
                       TOKEN=0;
                       END KEYWORD;
                       END LABALINIT;
                 KONIEC: PUT LIST ('DRUK SYMTAB I PRODUKCJI');
                       PUT DATA (STARTSYMB);
                          GOTO  KON;
                  BLAD1: PUT LIST ('TERMINAL PO LEWEJ STRONIE PRODUKCJI') SKIP;
                       GO TO KON;
                  BLAD2: PUT LIST ('BRAK ZNAKU = W  PRODUKCJI') SKIP;
                       GO TO KON;
```

```
   RCMAN: PROCEDURE OPTIONS(MAIN);

NEST

            BLAD3: PUT LIST ('ZA KROTKA SYNTAKS') SKIP;
                   GO TO ECM;
             BLAD4: PUT LIST ('BRAK DWUKROPKA PO NUMERZE PRODUKCJI') SKIP;
                   GO TO ECM;
             BLAD5: PUT LIST ('ZLA PRAWA STRONA PRODUKCJI') SKIP;
                   GO TO ECM;
             BLAD6: PUT LIST ('BRAK SYMBOLU STARTOWEGO') SKIP;
                   GO TO ECM;
                   /* ZAMIAST ECM DAC KONIEC PO WYRZUCENIU WYDRUKOW*/
               ECM:END RCMAN;
```

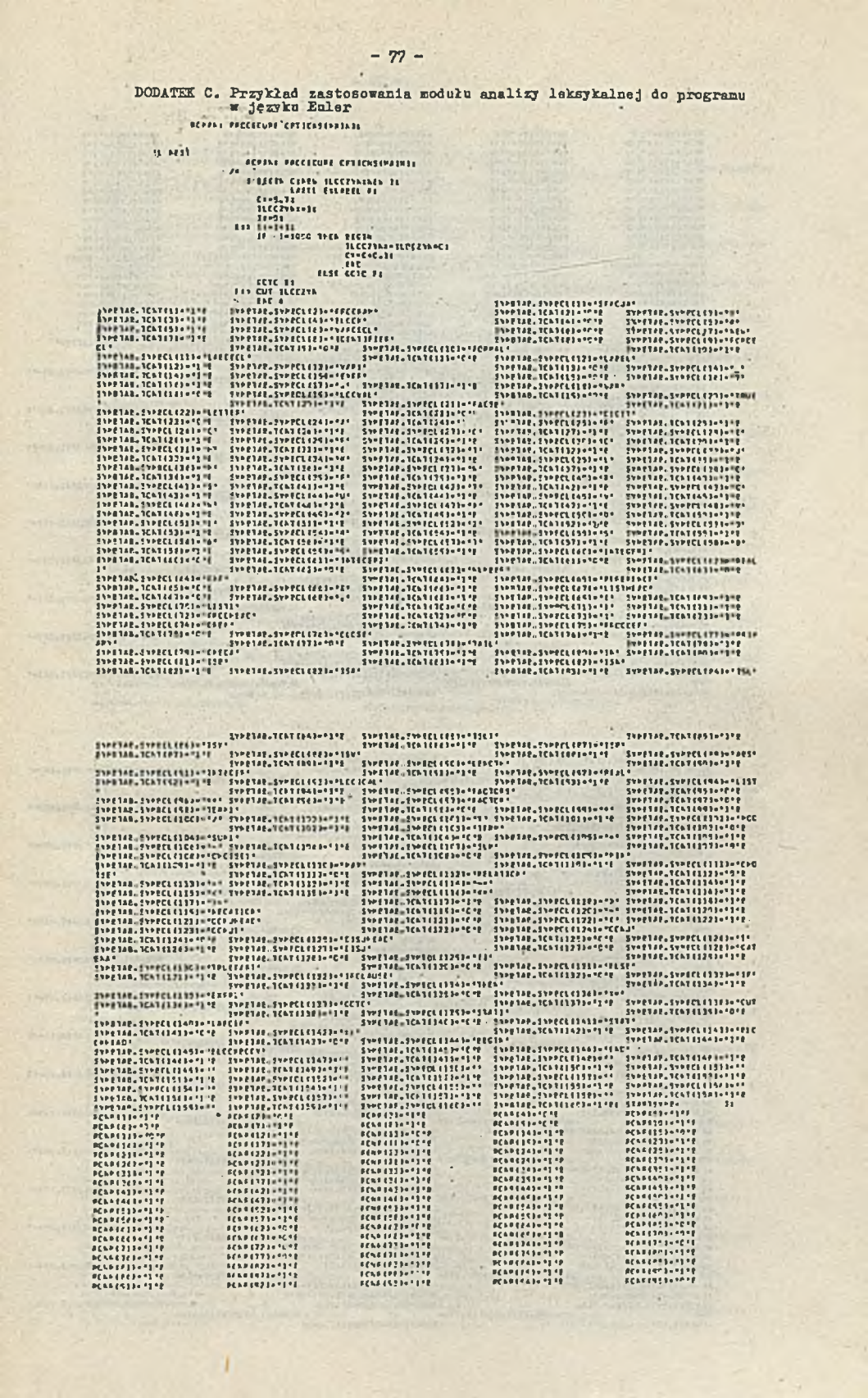
DODATEK B. Składnia języka Euler

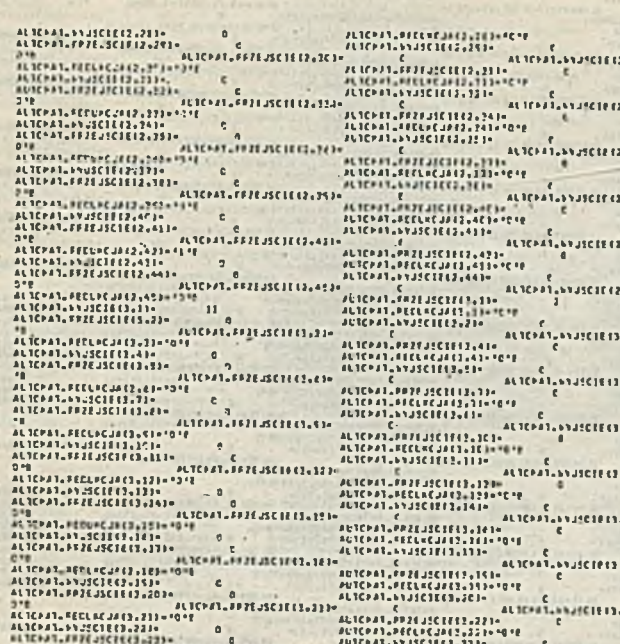
```
PROGRAM: PROCEDURE OPTIONS(MAIN);


TEST
          PROGRAM: PROCEDURE OPTIONS(MAIN);
        /*
          1:      PROGRAM     =     '$',BLOCK,'.';
          2:      VARDECL     =     'VAR',IDENTIFIER;
          3:      PROCDECL    =     'PROCEDURE',IDENTIFIER;
          4:      LABELDECL   =     'LABEL' ,IDENTIFIER;
          5:      VAR1        =     IDENTIFIER|
          6:                        VAR1,'_',EXPR,';'|
          7:                        VAR1,',';
          8:      VAR         =     VAR1;
          9:      LOGVAL      =     'TRUE'|
         10:                        'FALSE';
         11:      IDENTIFIER  =     LETTER|
         12:                        IDENTIFIER,LETTER|
         13:                        IDENTIFIER,DIGIT;
         14:      LETTER      =     'A'|
         15:                        'B'|
         16:                        'C'|
         17:                        'D'|
         18:                        'E'|
         19:                        'F'|
         20:                        'G'|
         21:                        'H'|
         22:                        'I'|
         23:                        'J'|
         24:                        'K'|
         25:                        'L'|
         26:                        'M'|
         27:                        'N'|
         28:                        'O'|
         29:                        'P'|
         30:                        'R'|
         31:                        'S'|
         32:                        'T'|
         33:                        'U'|
         34:                        'V'|
         35:                        'W'|
         36:                        'X'|
         37:                        'Y'|
         38:                        'Z';
         39:      DIGIT       =     '0'|
         40:                        '1'|
         41:                        '2'|
         42:                        '3'|
         43:                        '4'|
         44:                        '5'|
         45:                        '6'|
         46:                        '7'|
         47:                        '8'|
         48:                        '9';
         49:      INTEGER1    =     DIGIT|
         50:                        INTEGER1,DIGIT;
         51:      INTEGER2    =     INTEGER1;
         52:      REAL        =     INTEGER2,'.',INTEGER2|
         53:                        INTEGER2;
```

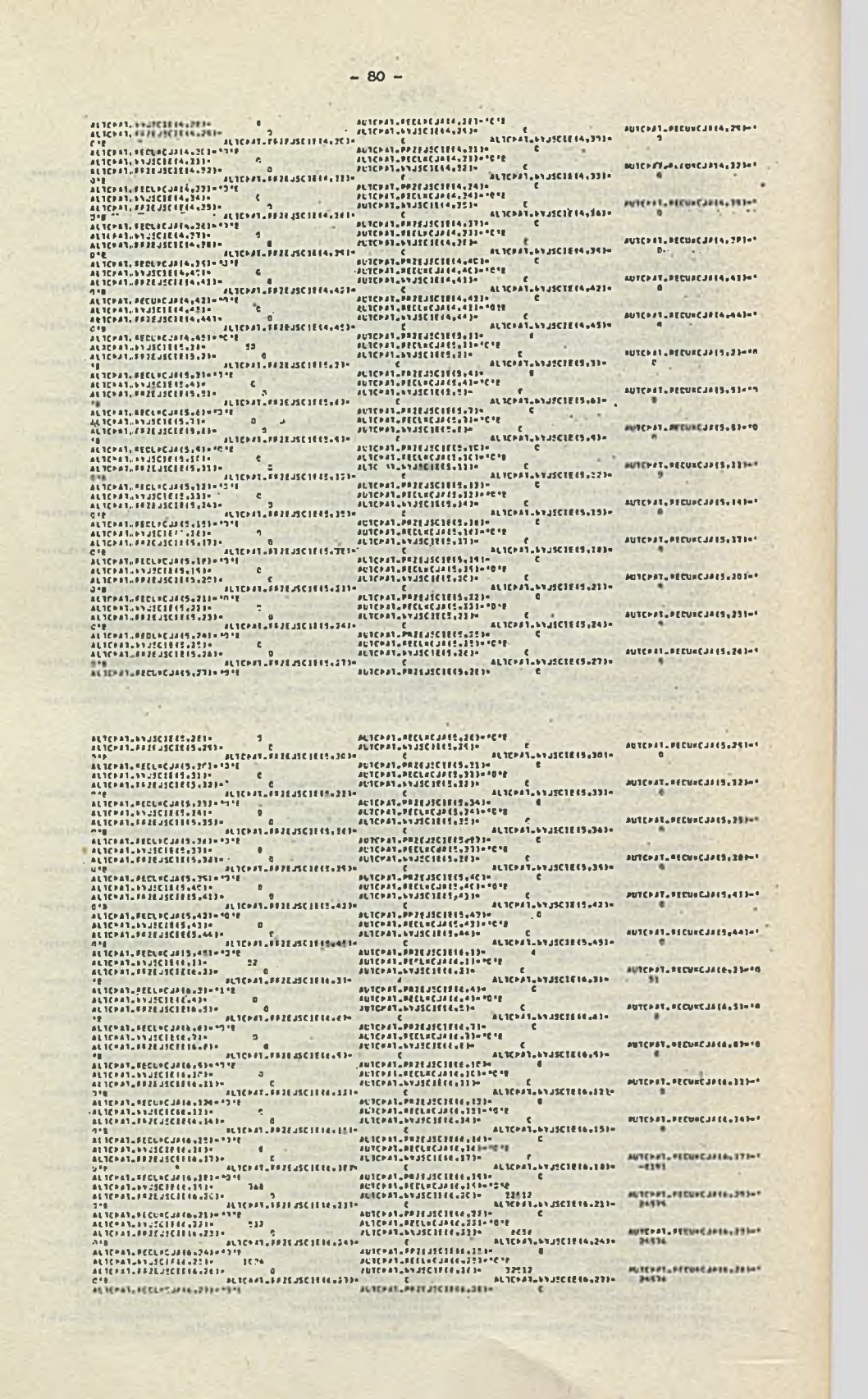
```
READ: PROCEDURE OPTIONS(MAIN);

TEST

          NUMBER     =    REAL|
                          REAL,'E>P',INTEGER2|
                          'E>P',INTEGER2;
          REFERENCE  =    '@',VAR;
          LISTHEAD   =    LISTHEAD,E>PR,',';
                          '(';
          LIST       =    LISTHEAD,E>PR,')';
                          LISTHEAD,')';
          PROCHEAD   =    PROCHEAD,PCPROCL,';'|
                          'OPEN';
          PROCDEF    =    PROCHEAD,E>PR,'CLOSE';
          PRIMARY    =    VAR|
                          VAR,LIST1|
                          LOCVAL|
                          NUMBER|
                          REFERENCE|
                          LIST1|
                          'TAIL',PRIMARY|
                          PROCDEF|
                          'OMEGA'|
                          '_',E>PR,'?'|
                          'IN'|
                          'ISB',VAR|
                          'ISA',VAR|
                          'ISR',VAR|
                          'ISL',VAR|
                          'ISL1',VAR|
                          'ISX',VAR|
                          'ISP',VAR|
                          'ISV',VAR|
                          'AES',PRIMARY|
                          'LENGTH',VAR|
                          'INTEGER',PRIMARY|
                          'REAL',PRIMARY|
                          'LOGICAL',PRIMARY|
                          'LIST',PRIMARY;
          FACTOR1    =    PRIMARY|
                          FACTOR1,'**',PRIMARY;
          FACTOR     =    FACTOR1;
          TERM1      =    FACTOR|
                          TERM1,'*',FACTOR|
                          TERM1,'/',FACTOR|
                          TERM1,'//',FACTOR|
                          TERM1,'MOD',FACTOR;
          TERM       =    TERM1;
          SUM1       =    TERM|
                          '+',TERM|
                          '-',TERM|
                          SUM1,'+',TERM|
                          SUM1,'-',TERM;
          SUM        =    SUM1;
          CHOISE1    =    SUM|
                          CHOISE1,'MIN',SUM|
                          CHOISE1,'MAX',SUM;
          CHOISE     =    CHOISE1;
```

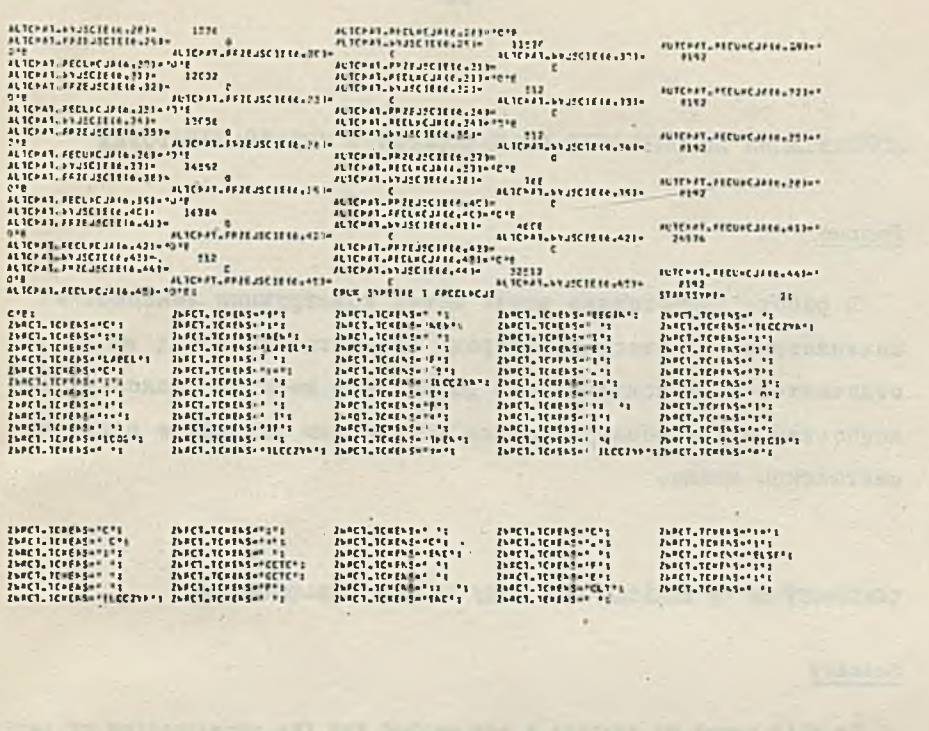
```
BOOL: PROCEDURE OPTIONS(MAIN);

TEST

          RELATION,  =    CHOISE|
                          CHOISE,'=',CHOISE|
                          CHOISE,'¬=',CHOISE|
                          CHOISE,'<',CHOISE|
                          CHOISE,'<=',CHOISE|
                          CHOISE,'>=',CHOISE|
                          CHOISE,'>',CHOISE;
          NEGATION   =    RELATION|
                          '¬',RELATION;
          CONJHEAD   =    NEGATION,'&';
          CONJ       =    CONJHEAD,CONJ|
                          NEGATION;
          CONJ       =    CONJ1;
          DISJHEAD   =    CONJ,'|';
          DISJ       =    DISJHEAD,DISJ|
                          CONJ;
          CATENA     =    CATENA,'||',PRIMARY|
                          DISJ;
          TRUEPART   =    EXPR,'ELSE';
          IFCLAUSE   =    'IF',EXPR,'THEN';
          EXPR1      =    BLOCK|
                          IFCLAUSE,TRUEPART,EXPR1|
                          VAR,':=',EXPR1|
                          'GOTO',PRIMARY|
                          'CLT',EXPR1|
                          CATENA;
          EXPR       =    EXPR1;
          STAT1      =    LABDEF,STAT1|
                          EXPR;
          STAT       =    STAT1;
          LABDEF     =    IDENTIFIER,':';
          BLOCKHEAD  =    'BEGIN'|
                          BLOCKHEAD,VARDECL,';'|
                          BLOCKHEAD,LABDECL,';';
          BLOCKBODY  =    BLOCKHEAD|
                          BLOCKBODY,STAT,';';
          BLOCK      =    BLOCKBODY,STAT,'END'.
          */
```

DODATEK C. Przykład zastosowania modułu analizy leksykalnej do programu w języku Euler









КОНСТРУКЦИЯ ЛЕКСИЧЕСКИХ АНАЛИЗАТОРОВ В МЕТАТРАНСЛЯТОРАХ

Резюме

В работе представлен новый метод конструкции лексических анализаторов в метатрансляторах. Предлагаемый метод на много отличается от использованных до сих пор методов поскольку лексические атомы обнаруживаются только лиш на основе описания синтаксиса языка.

CONSTRUCTION OF LEXICAL ANALYSERS IN COMPILER-COMPILERS

Summary

In this paper we present a new method for the construction of lexical analysers in compiler-compilers. The method presented is essentially different from those applied up to now since the lexical atoms are found automatically solely on the basis of the syntax description of the language.