ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1982

Seria: INFORMATYKA z. 3 Nr kol. 708

Jan BRUSKI

Politechnika Śląska
Ośrodek Elektronicznej
Techniki Obliczeniowej

KONCEPCJA DEFINIOWANIA SEMANTYKI BEZKONTEKSTOWYCH JĘZYKÓW PROGRAMOWANIA ZA POMOCĄ ABSTRAKCYJNEGO GENERATORA KODU

Streszczenie: W pracy przedstawiono koncepcję formalnego definiowania semantyki języków programowania za pomocą abstrakcyjnego generatora kodu i omówiono strukturę i istotę działania takiego generatora. Za pomocą schematów blokowych pokazano działanie podstawowego rozkazu abstrakcyjnego oraz kilku przykładowych rozkazów abstrakcyjnych, odpowiadających wybranym produkcjom pewnej gramatyki bezkontekstowej.
Przedstawiono również cel, jakiemu ma służyć proponowany sposób definiowania języków.

Teoriomnogościowa definicja języka opiera się na pojęciu gramatyki. Nie jest to jednak pełna definicja. Gramatyka określa bowiem jedynie składnię języka i pozwala na tworzenie ciągów symboli terminalnych, stanowiących poprawne wyrażenia języka. Produkcje gramatyki określają strukturę poprawnych wyrażeń języka, ale nie definiują ich znaczenia. Konieczne jest więc uzupełnienie:

Każdemu wyrażeniu poprawnemu danego języka trzeba przyporządkować pewne znaczenie, czyli treść semantyczną. Dotyczy to języków rozumianych w najszerszym tego słowa sensie, ale szczególnie jest istotne wtedy, gdy rozpatruje się języki programowania. Te bowiem języki służą celom kontaktu człowieka z maszyną cyfrową i dlatego muszą być zdefiniowane w sposób bardzo ścisły, nie zezwalający na dowolność interpretacji. Odnosi się to zarówno do syntaktyki języka, jak i do jego semantyki.

Syntaktyka języka programowania jest z reguły sformalizowana przez jego gramatykę, natomiast semantykę języka definiuje się zwykle w sposób opisowy za pomocą pewnego metajęzyka, którym jest zazwyczaj wybrany język etniczny. Formalizację semantyki języka programowania osiąga się dzięki translacji rozpatrywanego języka na język wewnętrzny maszyny cyfrowej.

W odniesieniu do konkretnej maszyny cyfrowej sposób ten jest pożyteczny, gdyż pozwala równocześnie na osiągnięcie celu, jakiemu służy język programowania. Musi jednak wtedy istnieć translator - specjalny program maszyny cyfrowej, tłumaczący poprawne wyrażenia języka programowania na rozkazy maszyny, a więc elementy wewnętrznego języka maszyny. Program ten jest z reguły opracowany w języku wewnętrznym danej maszyny cyfrowej lub w języku symbolicznym zbliżonym do języka wewnętrznego. Z tego względu jest on przydatny dla tego typu maszyny, w której języku został opracowany, natomiast jest zupełnie nieużyteczny dla innych typów maszyn cyfrowych.

Każdy typ maszyny cyfrowej wymaga odrębnego translatora danego języka, a opracowanie takiego translatora jest bardzo pracochłonne i czasochłonne. Fakt, że dla pewnej maszyny cyfrowej istnieje translator danego języka programowania wcale nie ułatwia zadania opracowania takiego translatora dla innej maszyny cyfrowej. Tworzenie translatora musi więc odbywać się na podstawie nieformalnego opisu semantyki języka, co powoduje, że zadanie to jest bardzo trudne, gdyż taki opis nie zawiera zazwyczaj żadnych wskazówek, jak należy budować translator.

Dodatkowo, korzystanie z nieformalnego opisu stwarza możliwość niejednoznacznej interpretacji treści semantycznych języka, a to może prowadzić do zbudowania translatora, którego działanie nie będzie poprawne.

Z przedstawionych powodów nie ustają wysiłki, aby opis semantyki języków programowania (niezależnie od opisu nieformalnego) można było zestawić w sposób sformalizowany, matematycznie ścisły i równocześnie tak, by ze sposobu tego wynikała bezpośrednio możliwość budowy translatora języka dla dowolnej maszyny cyfrowej.

Temu też celowi służy prezentowana praca, w której podjęto próbę zbudowania abstrakcyjnego generatora kodu, służącego formalizacji semantyki bezkontekstowych języków programowania.

W działaniu rzeczywistego translatora dowolnego języka programowania tłumaczącego programy przygotowane w tym języku można wyróżnić dwa podstawowe etapy: analizę syntaktyczną i generację kodu. Celem analizy syntaktycznej jest dokonanie rozbioru tłumaczonego programu i badanie składniowej poprawności tego programu. Wynikiem analizy syntaktycznej jest zazwyczaj zapisane w odpowiedniej postaci tzw. drzewo rozbioru programu.

Generacja kodu odbywa się w następnej fazie, po stwierdzeniu formalnej poprawności programu i polega na utworzeniu i umieszczeniu w pamięci maszyny cyfrowej tzw. programu wynikowego, stanowiącego ciąg rozkazów należących do listy rozkazów maszyny, uzupełnionych pewnymi parametrami zależnymi od struktury języka programowania i treści zawartych w programie źródłowym. Ta część translatora, który służy do generacji kodu i dlatego nazywana jest generatorem kodu, przyporządkowuje poszczególnym elementom języka zawartym w programie tłumaczonym ściśle określone sekwencje rozkazów.

Wykonanie tych sekwencji w pewnym ustalonym porządku przez procesor maszyny cyfrowej, a więc wykonanie rozkazów programu wynikowego powinno spowodować skutki przewidziane w programie źródłowym, a określone semantyką odpowiednich elementów języka programowania, które ten program źródłowy wykorzystuje.

Wynika stąd, że generator kodu może być również traktowany nieco inaczej, a mianowicie jako automat służący do definiowania semantyki języka programowania przy założeniu, że język maszyny cyfrowej, w którym opracowany jest translator, stanowi metajęzyk opisu semantyki.

Koncepcja definiowania semantyki bezkontekstowych języków programowania za pomocą abstrakcyjnego generatora kodu polega na wykorzystaniu tej ostatniej własności generatorów kodu. Można bowiem założyć istnienie pewnej abstrakcyjnej maszyny i dla tej maszyny zbudować abstrakcyjny translator, którego działanie może być zbliżone do działania rzeczywistego translatora w konkretnej maszynie cyfrowej. Abstrakcyjny generator kodu (będący częścią składową abstrakcyjnego translatora) będzie służył do tworzenia ciągów rozkazów abstrakcyjnych odpowiadających programom źródłowym opracowanym w określonym języku programowania. Generowane rozkazy abstrakcyjne będą równocześnie definiowały semantykę poszczególnych elementów języka programowania.

Rzeczywistą maszynę cyfrową można uważać za urządzenie zawierające pamięć, procesor oraz układ sterowania i współpracujące z pewnymi urządzeniami wejściowymi i wyjściowymi. Cechą charakterystyczną maszyny cyfrowej jest to, że może ona wykonywać pewną liczbę operacji określonych rozkazami należącymi do listy rozkazów tej maszyny. Zawartość listy rozkazów jest ściśle uzależniona od typu maszyny cyfrowej.

W podobny sposób zostanie określona maszyna abstrakcyjna. Przyjmuje się bowiem, że zawiera ona procesor abstrakcyjny oraz ośrodek przechowywania informacji. Cechą charakterystyczną abstrakcyjnego procesora jest to, że przesłanie do niego dowolnego rozkazu abstrakcyjnego powoduje natychmiastowe wykonanie tego rozkazu niezależnie od tego, jaki to rozkaz oraz niezależnie od tego czy odbywa się to w czasie procesu translacji programu (generacji kodu), czy też w trakcie wykonywania abstrakcyjnego programu wynikowego.

O ośrodku przechowywania informacji zakłada się, że posiada organizację strukturalną. Można w nim umieszczać dowolne obiekty będące strukturami danych. Można również wydzielać z niego pewne jego fragmenty i traktować jako oddzielne układy przechowywania informacji. Poza nielicznymi wyjątkami nie określa się wtedy pojemności takich układów.

Układy przechowywania informacji dzieli się na dwie grupy: układy translatora abstrakcyjnego (abstrakcyjnego generatora kodu) i układy abstrakcyjnego programu wynikowego. Podział ten nie stwarza rozłącznych zbiorów układów przechowywania informacji, gdyż niektóre układy mogą być wykorzy-

stywane zarówno w procesie generacji kodu, jak i w trakcie wykonywania programu wynikowego.

Dla maszyny abstrakcyjnej nie określa się z góry listy rozkazów. Zakłada się natomiast, że każdy dowolny rozkaz może być wykonany przez jej procesor.

Rozkazy abstrakcyjne związane są z językiem programowania, dla potrzeb którego buduje się abstrakcyjny translator. Liczba abstrakcyjnych rozkazów zależy od składni języka, natomiast ich treść - od semantyki tego języka.

Zakłada się, że działanie abstrakcyjnego translatora a ściślej biorąc, abstrakcyjnego generatora kodu sterowane jest składnią rozpatrywanego języka programowania.

Rozkazy abstrakcyjne dzieli się na dwie grupy: rozkazy translatora oraz rozkazy programu wynikowego.

Zadaniem poszczególnych rozkazów abstrakcyjnych translatora jest tworzenie odpowiednich obiektów w ośrodku przechowywania informacji, a wśród nich tworzenie zbioru wyjściowego translatora. Odpowiednie rozkazy translatora wprowadzają do tego zbioru rozkazy abstrakcyjne programu wynikowego i ewentualnie parametry uzupełniające te rozkazy.
Proces ten może przebiegać sukcesywnie, sterowany składnią programu źródłowego. Po zakończeniu generacji kodu zawartość zbioru wyjściowego translatora będzie stanowiła abstrakcyjny program wynikowy. Wykonanie kolejnych rozkazów tego programu wywoła skutki określone semantyką rozpatrywanego języka programowania.

Każdej produkcji gramatyki języka programowania przyporządkowana jest jedna sekwencja rozkazów abstrakcyjnych translatora (najczęściej będzie to jeden rozkaz abstrakcyjny). Oprócz tego zbiór rozkazów abstrakcyjnych zawiera kilka rozkazów o charakterze organizacyjnym.

Poszczególnym produkcjom gramatyki języka (ale nie wszystkim) przyporządkowuje się również rozkazy abstrakcyjne programu wynikowego (znów najczęściej jednej produkcji odpowiada jeden rozkaz abstrakcyjny). Zbiór rozkazów abstrakcyjnych programu wynikowego jest także uzupełniony pewnymi rozkazami organizacyjnymi.

Abstrakcyjny generator kodu działa w ten sposób, że rozpoznaje kolejne produkcje odpowiadające poszczególnym zapisom w programie źródłowym, wybiera rozkazy abstrakcyjne translatora przyporządkowane tym produkcjom i przesyła je do procesora abstrakcyjnego, gdzie następuje ich wykonanie. Funkcje te spełnia specjalny rozkaz organizacyjny translatora nazwany rozkazem podstawowym. Pozostałe rozkazy abstrakcyjne translatora określają pewne operacje na obiektach w ośrodku przechowywania informacji. Niektóre rozkazy abstrakcyjne translatora umieszczają w zbiorze wyjściowym translatora odpowiednie rozkazy abstrakcyjne programu wynikowego.

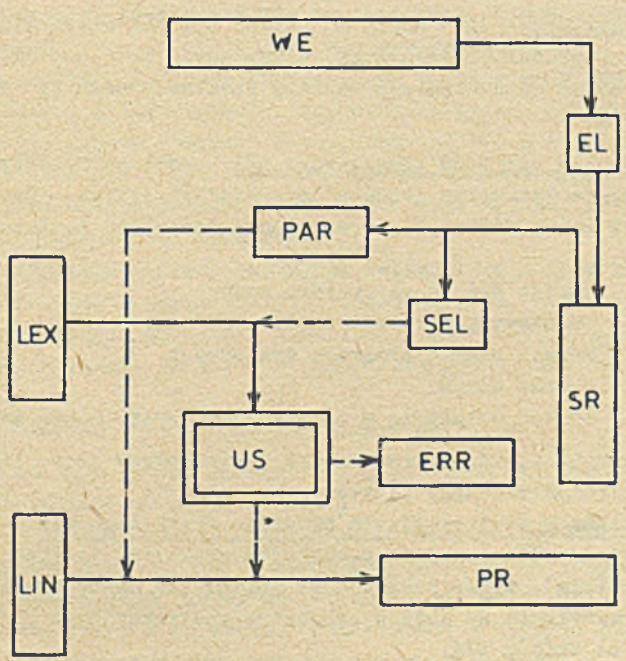
Sposób działania podstawowego rozkazu translatora zależy od tego, w jakiej postaci są przygotowane informacje wejściowe.

Dla uproszczenia przyjęto, że rozpatruje się programy syntaktycznie poprawne po dokonaniu analizy składniowej, a więc po uzyskaniu drzewa rozbioru.

Zakłada się również, że otrzymane drzewo rozbioru przedstawione jest w postaci liniowej dzięki przekształceniu gramatyki języka do gramatyki pochodnej. Jeżeli gramatyka bezkontekstowego języka programowania jest dana jako $G = (T, N, P, S)$, gdzie T jest alfabetem symboli terminalnych, N - alfabetem symboli nieterminalnych, P - zbiorem produkcji, a S - aksjomatem gramatyki, to gramatyka pochodna jest zdefiniowana w postaci:

$$G' = (T', N, P', S) \quad \text{gdzie:}$$
$$T' = T \cup \{A[\,|\,A \in N\} \cup \{]\}$$
$$P' = \{A \to A[\alpha] \,|\, (A \to \alpha) \in P\}.$$

Zadaniem abstrakcyjnego generatora kodu jest analiza symboli zawartych w zapisie drzewa rozbioru i utworzenie na tej podstawie ciągu rozkazów abstrakcyjnych programu wynikowego, odzwierciedlających znaczenie poszczególnych elementów programu źródłowego.



Rys. 1. Schemat organizacyjny abstrakcyjnego generatora kodu

Podstawowe układy abstrakcyjnego generatora kodu, przy przyjętych założeniach posiadają niezmienny charakter i strukturę, niezależnie od języka programowania i mogą być zawsze jednakowe dla wszystkich języków bezkontekstowych. Układy te oraz ich powiązania przedstawiono na rys. 1.

Zbiór wejściowy WE zawiera liniowo uporządkowany zapis drzewa wywodu otrzymanego w wyniku analizy syntaktycznej. Ze zbioru tego odczytuje się kolejno zapisane tam symbole, natomiast nie zapisuje się już w tym zbiorze żadnych informacji.

Stos roboczy generatora kodu SR. Przepisuje się do niego kolejne symbole ze zbioru WE, a w momencie zidentyfikowania symbolu ⟧ odczytuje się z niego wszystkie kolejne symbole, aż do natrafienia symbolu ⟦ włącznie. Każdy zapis do stosu SR zwiększa jego pojemność, natomiast odczyt ją zmniejsza.

Układ EL - służy jako pośredni element przechowywania symboli odczytanych ze zbioru WE przed ich wpisaniem do stosu SR.

Stos parametrów PAR - służy do zapisu symboli odczytanych ze stosu SR, będących składnikami produkcji gramatyki. Głębokość stosu PAR jest ograniczona maksymalną liczbą składników występujących w produkcjach gramatyki rozpatrywanego języka.

Magazyn selektorów SEL - służy do zapisu symboli nieterminalnych, stanowiących nazwy produkcji gramatyki, odczytanych ze stosu SR. Symbole te decydują o wyborze odpowiedniego rozkazu abstrakcyjnego ze zbioru rozkazów translatora.

Zbiór (słownik) rozkazów translatora LEX - zawiera rozkazy abstrakcyjne, odpowiadające wszystkim produkcjom gramatyki języka. Oprócz nich w skład słownika rozkazów translatora wchodzą dodatkowo następujące rozkazy abstrakcyjne:

ex - sttr rozkaz startu generatora kodu
ex - ptr podstawowy rozkaz translatora
ex - stpr rozkaz startu programu wynikowego
ex - stop rozkaz stopu
ex - err rozkaz działania w przypadku wykrycia błędu semantycznego.

Pojemność słownika rozkazów translatora uwarunkowana jest liczbą produkcji gramatyki rozpatrywanego języka.

Słownik rozkazów abstrakcyjnych programu wynikowego LIN - zawiera rozkazy abstrakcyjne odpowiadające produkcjom gramatyki języka (ale niekoniecznie wszystkim). Wybór rozkazów ze zbioru LIN następuje w trakcie wykonywania (przesyłania do układu procesora abstrakcyjnego US) rozkazów abstrakcyjnych ze zbioru LEX.

Oprócz rozkazów odpowiadających poszczególnym produkcjom gramatyki słownik LIN dodatkowo zawiera następujące rozkazy abstrakcyjne:

int - stpr rozkaz startu programu wynikowego
int - stop rozkaz stopu
int - err rozkaz działania w przypadku wykrycia błędu podczas wykonywania programu.

Pojemność zbioru LIN uzależniona jest od liczby efektywnych produkcji gramatyki języka.

Program wynikowy PR jest zbiorem wyjściowym abstrakcyjnego generatora kodu. Do zbioru tego dokonuje się zapisu rozkazów abstrakcyjnych pochodzących ze zbioru LIN z ustalonymi stanami początkowymi. Stany początkowe określa generator kodu za pomocą rozkazów abstrakcyjnych ze zbioru LEX. W miarę zapisu kolejnych rozkazów abstrakcyjnych pojemność zbioru PR stopniowo rośnie, osiągając ekstremum w chwili zakończenia procesu translacji.

Zbiór błędów semantycznych programu wynikowego ERR - służy do zapisu informacji specyfikujących ewentualne błędy semantyczne występujące w programie źródłowym. Wymagana pojemność zbioru ERR zależy od liczby błędów semantycznych istniejących w programie.

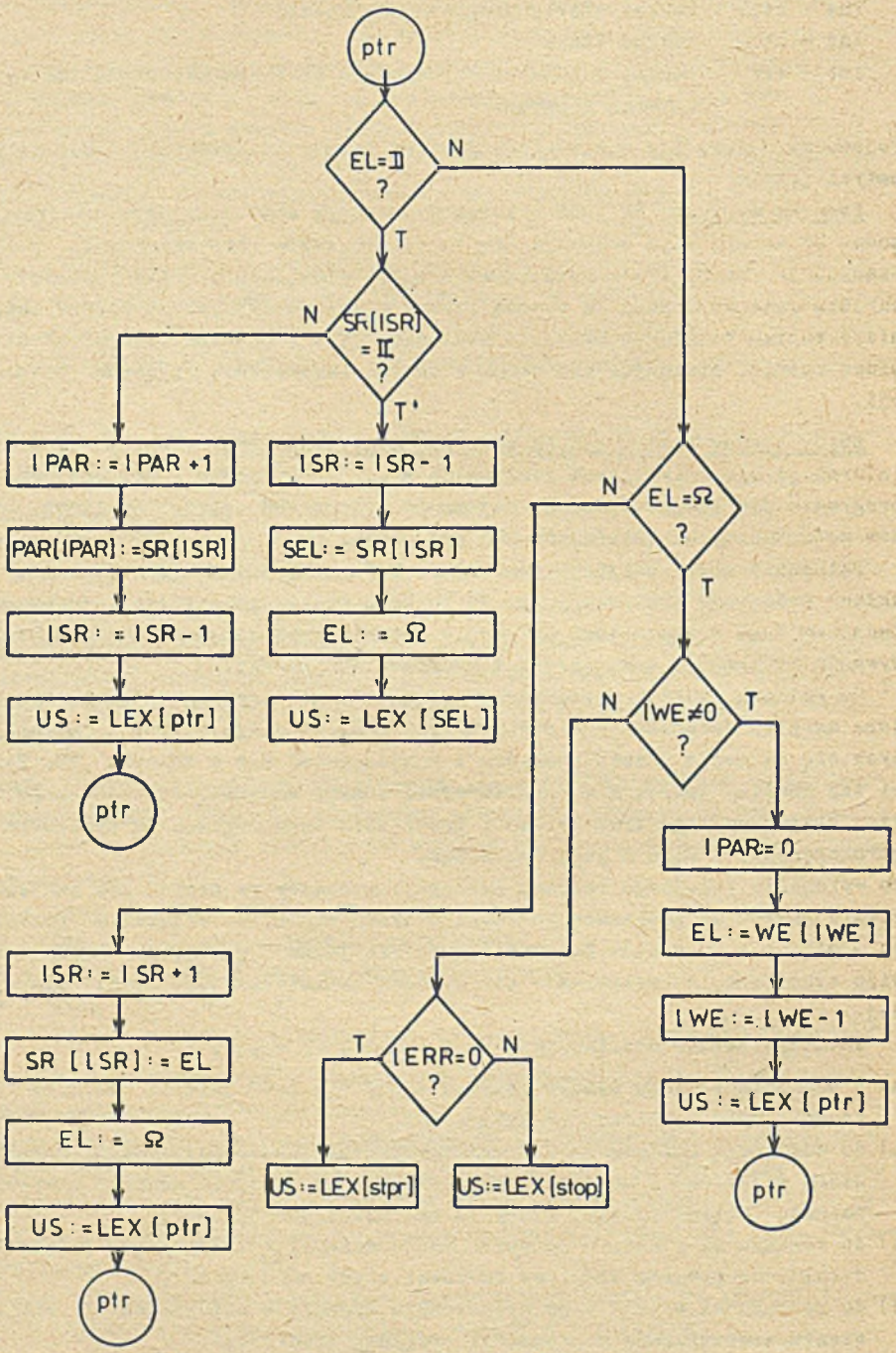
Działanie abstrakcyjnego generatora kodu rozpoczyna się przesłaniem do układu procesora abstrakcyjnego US rozkazu startu generatora kodu ex-sttr. Zadaniem tego rozkazu jest ustalenie początkowych stanów układów przechowywania informacji translatora i programu wynikowego.

Od rozkazu ex-sttr następuje przejście do podstawowego rozkazu translatora ex-ptr. Zadaniem tego rozkazu jest analiza ciągu symboli składających się na zapis drzewa rozbioru i znajdujących się w zbiorze WE. Wyniki tej analizy decydują o wyborze przez rozkaz ex - ptr odpowiedniego rozkazu abstrakcyjnego translatora i przesłaniu tego rozkazu do procesora abstrakcyjnego, a więc o jego wykonaniu.

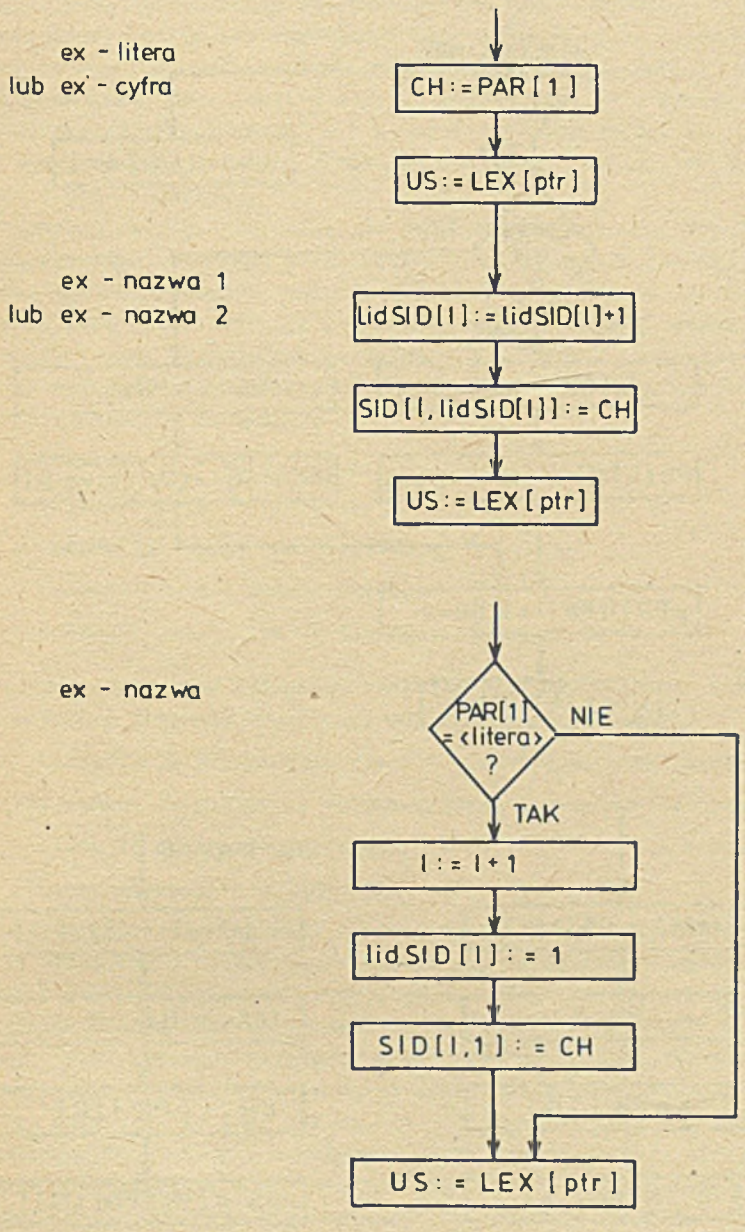
Po wykonaniu kolejnego rozkazu lub grupy rozkazów ze zbioru LEX następuje zawsze powrót do podstawowego rozkazu translatora (z wyjątkiem rozkazów ex - stpr i ex - stop). Działanie abstrakcyjnego generatora kodu jest więc cykliczne, a rozkaz ex - ptr stanowi zamknięcie jego pętli sterującej.

Istnieją cztery możliwe wyjścia z rozkazu ex - ptr;

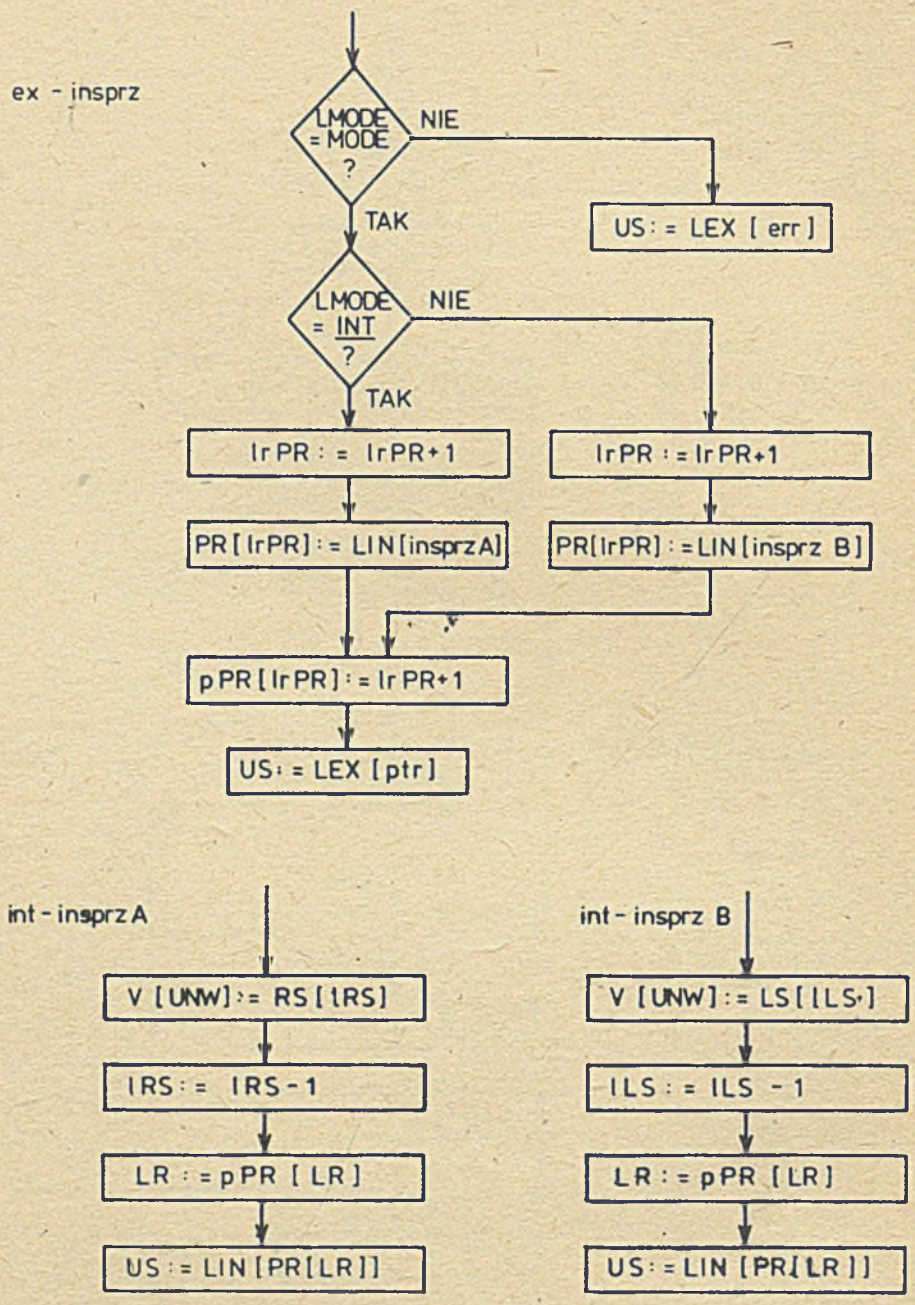
a) z powrotem do tego samego rozkazu - gdy nie jest jeszcze możliwe dokonanie wyboru rozkazu ze zbioru LEX,
b) do dowolnego rozkazu abstrakcyjnego translatora - gdy wyniki analizy ciągu wejściowego ze zbioru WE zezwalają na podjęcie decyzji o wyborze rozkazu abstrakcyjnego, który ma być aktualnie wykonany,
c) do rozkazu ex - stop - po wyczerpaniu symboli w zbiorze WE i stwierdzeniu, że program źródłowy zawierał błędy semantyczne,
d) do rozkazu ex - stpr - po wyczerpaniu symboli w zbiorze WE i stwierdzeniu semantycznej poprawności programu źródłowego.



Rys. 2. Schemat blokowy podstawowego rozkazu abstrakcyjnego translatora



Rys. 3. Schematy blokowe rozkazów abstrakcyjnych kompletujących nazwy



Rys. 4. Schematy blokowe rozkazów abstrakcyjnych odpowiadających instrukcji przypisania

Działanie podstawowego rozkazu translatora zostało przedstawione za pomocą schematu blokowego na rys. 2. (Do zapisu poszczególnych operacji użyto tu notacji zbliżonej do stosowanej w języku ALGOL).

Podstawowy rozkaz abstrakcyjnego generatora kodu ex - ptr wykorzystuje zawsze te same układy przechowywania informacji. Pozostałe rozkazy abstrakcyjne korzystają również z niektórych z nich, ale wymagają także i innych. Liczba tych układów, ich struktura i właściwości są zależne od języka programowania i bardzo ściśle są związane z syntaktyką języka i oczywiście z jego semantyką.

W podobny sposób, jak działanie podstawowego rozkazu translatora, za pomocą schematu blokowego można przedstawić działanie każdego dowolnego rozkazu abstrakcyjnego ze zbiorów LEX i LIN. Rozkazy te trzeba jednak rozpatrywać łącznie i wiązać je z konkretnymi produkcjami gramatyki rozpatrywanego języka programowania. Na rys. 3 pokazano przykładowo schematy blokowe rozkazów abstrakcyjnych translatora, które mogą odpowiadać następującym produkcjom gramatyki:

<litera> :: = A |B| ..... |Z

<cyfra> :: = 0 |1| ..... |9

<nazwa 1>:: = <nazwa> <litera>

<nazwa 2>:: = <nazwa> <cyfra>

<nazwa> :: = <litera >|<nazwa 1>|< nazwa 2>

Na rys. 4 pokazano natomiast schematy blokowe rozkazów abstrakcyjnych translatora i programu wynikowego odpowiadające produkcji:

<instrukcja przypisania >:: = <lewa strona> <wyrażenie>

przy założeniu, że gramatyka rozpatrywanego języka zawiera produkcje:

<lewa strona> :: = <zmienna> : =

<typ zmiennej>:: = INT | LOG

Wszystkie obiekty tworzone i przetwarzane w procesie translacji lub w czasie wykonywania programu wynikowego należą do zbioru ogólnych struktur danych i stanowią klasy struktur danych, które określa się za pomocą odpowiednich predykatów. Formalnego zapisu takich obiektów można dokonać stosując tzw. notację wiedeńską. Notacja ta pozwala na określenie wszystkich używanych obiektów, a także na zapis operacji związanych z przetwarzaniem obiektów. Do tego ostatniego celu wykorzystuje się funkcje wyboru i konstrukcji obiektów i z sekwencji tych funkcji buduje się poszczególne rozkazy abstrakcyjne. W zapisie rozkazów abstrakcyjnych mogą również występować pewne operacje algebraiczne lub logiczne dotyczące obiektów elementarnych. Zakłada się, że semantyka tych operacji jest oczywista - przestrzega się przy tym reguł odpowiedniej algebry.

Stosowanie notacji wiedeńskiej stwarza pewne podobieństwo przedstawionej koncepcji z tzw. wiedeńską metodą definiowania języków za pomocą automatów interpretacyjnych. Między tymi dwoma sposobami istnieje jednak różnica jakościowa polegająca na innym działaniu automatu abstrakcyjnego; w metodzie wiedeńskiej jest to działanie interpretacyjne, natomiast w proponowanej metodzie - działanie kompilacyjne.

Zapisy rozkazów abstrakcyjnych w przedstawionej metodzie stanowią metajęzyk opisu semantyki rozpatrywanego języka programowania. Przyjęty sposób działania abstrakcyjnego generatora kodu oraz sposób tworzenia rozkazów abstrakcyjnych powodują, że uzyskuje się pewien model rzeczywistego translatora.

Sformułowanie rozkazów abstrakcyjnych, a więc sformalizowanie semantyki elementów języka nie jest oczywiście łatwe, ale jest to zadanie bez porównania prostsze w realizacji niż budowanie rzeczywistego generatora kodu na podstawie słownego opisu semantyki języka. Dzięki temu, że maszyna abstrakcyjna została bardzo ogólnie określona, raz zbudowany abstrakcyjny generator kodu pozwala na skuteczne i łatwe opracowywanie translatorów języka dla dowolnych maszyn cyfrowych. W trakcie tej czynności zasadniczym problemem pozostaje jedynie optymalizacja ich działania.

LITERATURA

[1] AHO A.V., ULLMAN J.D.: The Theory of Parsing, Translation and Campiling. Prentice - Hall, Inc. Englewood Cliffs, 1972.

[2] DIJKSTRA E.W.: Recursive programming Num. Math. 2, 1960.

[3] GOOS G., HARTMANIS J.: Methods of Algorithmic Language Implementation. Springer - Verlag. Berlin 1977.

[4] LEE J.A.N.: Computer Semantics. Van Nostrand Reinhold. New York 1972.

[5] LUCAS P.: Formal Definition of Programming Languages and Systems - IFIP Congress 1971.

[6] OLLONGREN A.: Definition of Programming Languages by Interpreting Automata.

[7] WEGNER P.: The Vienna Definition Language. A.C.M. Computing Surveys 4. Nr 1. March 1972.

Wpłynęło do Redakcji: 12.04.1979 r.
W ostatecznej formie przyjęto: 6.01.1981 r.

Recenzent: Doc.dr inż. Romuald Marczyński

КОНЦЕПЦИЯ ОПРЕДЕЛЕНИЯ СЕМАНТИКИ КОНТЕКСТНО-СВОБОДНЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ ПРИ ПОМОЩИ АБСТРАКТНОГО ГЕНЕРАТОРА КОДА

Р е з ю м е

Статья содержит концепцию формального определения семантики языков программирования при помощи абстрактного генератора кода, а также обсуждена структура и сущность работы этого генератора. С помощью блоков-схем представлены действие основной абстрактной команды, а также несколько примерных абстрактных команд соответствующих выбранным правилам некоторой контекстно-свободной грамматики. Представлена тоже цель, для которой предназначен предлагаемый способ определения языков.

A CONCEPT OF DEFINING SEMANTICS OF CONTEXT-FREE PROGRAMMING LANGUAGES BY MEANS OF ABSTRACT CODE GENERATOR

S u m m a r y

The paper discusses formal defining of semantics of context-free programming languages by means of abstract code generator, as well as the structure and essence of operation of such a generator. The operation of a basic abstract command, and few exemplary abstract commands representing the chosen productions of some context-free grammar has been shown by means of flow-charts.

The aim of the proposed manner of defining languages has been also presented.