

INSTYTUT CYBERNETYKI TECHNICZNEJ
POLITECHNIKI WROCŁAWSKIEJ

Raport serii: PREPRINTY nr 61/93

REPREZENTACJA WIEDZY WYRAŻONEJ
W ZDANIACH JĘZYKA NATURALNEGO
Z UWZGLĘDNIENIEM SEMANTYKI POJĘĆ
I MOŻLIWOŚCI DEDUKCJI

(praca doktorska)

Olgierd UNOLD

PROMOTOR:

prof. dr hab. inż. Jan Kazimierczak

Słowa kluczowe:

- sztuczna inteligencja
- reprezentacja wiedzy
- przetwarzanie języka naturalnego
- automat ze stosem

Wydawnictwo Politechniki Wrocławskiej
WROCŁAW 1993

SPIS TREŚCI

1. WPROWADZENIE	1
1.1. Klasyfikacja NLP systemów.....	1
1.2. Lingwistyczne problemy przetwarzania wiedzy wyrażonej w języku naturalnym.....	3
1.3. Przetwarzanie języka naturalnego.....	4
1.3.1. Słowniki	4
1.3.2. Analiza syntaktyczna.....	5
1.3.3. Analiza semantyczna.....	6
1.3.4. Reprezentacja wiedzy.....	7
1.3.5. Generacja wypowiedzi	8
1.4. Cel i zakres pracy.....	9
2. DEKOMPOZYCJA ZDANIA NA ELEMENTY BAZY WIEDZY.....	11
2.1. Dekompozycja zdania na komponenty strukturalno-semantyczne	11
2.2. Dekompozycja zdania na grupy słów.....	12
2.3. Model formalny automatu dekomponującego ciąg (w) na podciągi (w) _i	13
2.3.1. Struktura automatu	15
2.3.2. Definicja formalna automatu	17
2.3.3. Instrukcje automatu.....	18
2.3.4. Algorytm działania automatu.....	21
2.3.5. Graf przejść automatu	22
2.3.6. Dekompozycja ciągu (w) na podciągi (w) _i	30
2.4. Realizacja programowa automatu dekomponującego ciąg (w) na podciągi (w) _i	36
2.5. Opis funkcjonalny programu AUTOMAT	42
2.5.1. Opcja ≡.....	43
2.5.2. Opcja Zdanie.....	44
2.5.3. Opcja Podział.....	45
2.5.4. Opcja Okno	46
2.5.5. Opcja Wyjście	47
2.6. Perspektywy rozwoju i realizacji automatu <i>PDAMS</i>	47
3. FORMALNY SYSTEM REPREZENTACJI WIEDZY WYRAŻONEJ W ZDANIU JĘZYKA NATURALNEGO.....	49
3.1. Formalna reprezentacja komponentu o strukturze grupy werbalnej osobowej.....	49
3.1.1. Preliminaria gramatyczne	49

3.1.2. Analiza gramatyczna komponentu	50
3.1.3. Synteza gramatyczna komponentu	50
3.1.4. Przesłanki reprezentacji komponentu	51
3.1.5. Formalna reprezentacja komponentu	52
3.2. Transformacja wiedzy wyrażonej w zdaniu języka naturalnego na postać symboliczną	53
3.3. Synteza wiedzy w bazie wiedzy	59
3.4. Odtwarzanie wiedzy z bazy wiedzy	63
4. PROPOZYCJA REPREZENTACJI WIEDZY DLA POTRZEB ANALIZY SEMANTYCZNEJ	68
4.1. Zbiór kategorii semantycznych	68
4.1.1. Kategorie semantyczne procesu materialnego	70
4.1.2. Kategorie semantyczne procesu myślowego	70
4.1.3. Kategorie semantyczne procesu werbalnego	70
4.1.4. Kategorie semantyczne procesu atrybutowego	71
4.1.5. Kategorie semantyczne procesu identyfikującego	71
4.1.6. Kategorie semantyczne okoliczności towarzyszących procesowi	71
4.2. Formalna reprezentacja atrybutów semantycznych elementów bazy wiedzy	71
4.3. Możliwości przeprowadzenia wnioskowania proceduralnego	72
5. PODSUMOWANIE	75
LITERATURA	76
SKOROWIDZ SYMBOLI I OZNACZEŃ	85
ZAŁĄCZNIK A	88
ZAŁĄCZNIK B	123
ZAŁĄCZNIK C	127

1. WPROWADZENIE

Celem, ogłoszonego pod koniec lat 80-tych, amerykańskiego projektu komputerów szóstej generacji jest skonstruowanie maszyny w określonym stopniu wzorowanej na ludzkim mózgu i zdolnej do różnego rodzaju interakcji z otoczeniem, w tym również porozumiewania się z człowiekiem w języku naturalnym [Arbib, 1989]. Wyposażenie systemu komputerowego w możliwość komunikowania się z człowiekiem w języku naturalnym spełnia oczekiwania masowego użytkownika, który najczęściej nie posiada przygotowania informatycznego.

Prace nad automatycznym przetwarzaniem języka naturalnego (ang. natural language processing - NLP lub computational linguistics) prowadzone są od początku lat 50-tych. Jednak postęp w tej dziedzinie dokonuje się stosunkowo wolno. Zbudowanie systemu potrafiącego prowadzić dialog z użytkownikiem w języku naturalnym wymaga rozwiązania bardzo wielu problemów i to na gruncie różnych dyscyplin naukowych: lingwistyki, psycholingwistyki, sztucznej inteligencji. Sam człowiek potrzebuje lat nauki i treningu na to, by opanować słownictwo, gramatykę, znaczenie poszczególnych wyrazów czy jednostek frazeologicznych, wreszcie udoskonalić metody doboru środków wyrazu. Skonstruowanie systemu z dostępem w języku naturalnym (NLP systemu) wymaga więc udostępnienia systemowi możliwie pełnej wiedzy na temat danego języka, co wiąże się z wieloma trudnościami. Z drugiej strony, pomimo istnienia bogatej literatury lingwistycznej, ciągle brak jest teorii lingwistycznych opisujących całość problemów związanych z językiem naturalnym w sposób dogodny dla obliczeń automatycznych. Większość prac językoznawczych ma charakter niesformalizowany, czysto opisowy, natomiast istniejące formalne modele lingwistyczne nie obejmują wszystkich cech języka, a najbardziej ogólne z nich napotykają na poważne trudności w komputerowej realizacji¹.

Z tego względu wszystkie budowane obecnie NLP systemy realizują zadanie komunikacji w języku naturalnym wciąż w bardzo ograniczonym zakresie.

1.1. Klasyfikacja NLP systemów

W zależności od przeznaczenia NLP systemów można dokonać następującej ich klasyfikacji:

¹ Istniejące obecnie NLP systemy to albo systemy eksperymentalne, weryfikujące postawione założenia teoretyczne, albo proste systemy użytkowe, w których wykorzystuje się ściśle określone funkcje języka naturalnego.

○ systemy tłumaczące: ich zadaniem jest tłumaczenie tekstu z jednego języka naturalnego na inny (lub inne); przykładem są systemy TAUM_METEO [Thouin, 1982], TAUM_AVIATION [Isabelle, Bourbeau, 1985], TITUS 4 [Ducrot, 1982], system Weidnera [Hundt, 1982], METAL [Bennett, Slocum, 1985], SUSY [Luckhardt, 1982].

○ systemy dostępu do baz danych: przetwarzają pytania zadane w języku naturalnym na zdania języka zapytań bazy (np. SQL). Odpowiedzi systemu mogą być podane w postaci zdań języka naturalnego; przykładem są systemy LIFER [Hendrix i inni, 1978], PLANES [Waltz, 1978], RENDEZVOUS [Codd, 1978], INTELLECT [1980], ASK [Thompson, Thompson, 1985], IR-NLI [Guida, Tasso, 1983], TEAM [Grosz, 1983], CO-OP [Kaplan, 1984], TEXT [McKeown, 1985], IRUS [Bates, Moser, Stallard, 1986], KID [Ishikawa, 1987].

○ systemy interpretacyjne: umożliwiają analizę wewnętrznej struktury i zawartości znaczeniowej tekstu - system EPISTALE [Heidorn i inni, 1982], [Durham, Lamb, Saxe, 1983], [Ram, 1992]; umożliwiają sortowanie dokumentów i automatyczne ich streszczanie - [Paice, 1981], system FRUMP [DeJong, 1982], MOPS [Schank, 1982], [Kolodner, 1983].

○ systemy akwizycji i prezentacji wiedzy: systemy tego typu odczytują z tekstu zapisanego w języku naturalnym wiedzę, tworzą jej wewnętrzną reprezentację i zapamiętują w bazie wiedzy, w celu późniejszego jej wykorzystania (np. udzielając odpowiedzi na pytanie); przykładem są systemy REL [Thompson, Thompson, 1975], USL [Blaser, 1988], [Kazimierczak, 1990].

○ systemy bezpośredniego porozumiewania się: należą do multimedialnych, najbardziej zaawansowanych NLP systemów. Ich zadaniem jest najczęściej analiza ustnej wypowiedzi, jej przetworzenie językowe, wreszcie przetłumaczenie odpowiedzi na język docelowy i synteza do postaci dźwięku w tym języku; opis systemów multimedialnych można znaleźć w pracach [Conklin, 1987] i [Nielsen, 1990].

Podstawą powyższej klasyfikacji NLP systemów są ich cechy funkcjonalne (zob. [Salton, 1989]). Innym kryterium klasyfikacji może być przyjęty podczas tworzenia NLP systemu model poznawczy. Według tego kryterium można wyróżnić dwa typy systemów:

○ systemy działające w oparciu o paradygmat symboliczny: w systemach tych zakłada się, że wyższe funkcje umysłowe, a co za tym idzie i przetwarzanie języka naturalnego to manipulacja symbolami. Zgodnie z tym paradygmatem działa większość istniejących

obecnie NLP systemów, również o architekturze równoległej, rozproszonej² (np. [Fahlman, 1979], [Bandyopadhyay, 1991]).

○ systemy działające w oparciu o paradygmat konekcyjnistyczny: w systemach tych w przetwarzaniu języka naturalnego wykorzystuje się sztuczne sieci neuronowe; przykładem neuronowego przetwarzania języka naturalnego są systemy SDM [Kanerva, 1988], DISCERN [Mäikkiläinen, Dyer, 1990], DETE [Nenov, 1991].

1.2. Lingwistyczne problemy przetwarzania wiedzy wyrażonej w języku naturalnym

Wyrażenie wiedzy w języku naturalnym jest dla użytkownika-człowieka NLP systemu sytuacją niezwykle dogodną, lecz jest jednocześnie źródłem niemal lawinowo pojawiających się problemów związanych z automatycznym przetwarzaniem tak elastycznego, bogatego i niejednoznacznego tworu, jakim jest język naturalny. W rozdziale zostaną przedstawione tylko niektóre lingwistyczne problemy, jakie pojawiają się w procesie automatycznego przetwarzania języka naturalnego (NLP procesie):

- problem wieloznaczności na poziomie słów:

Czy w zdaniu *They can fish* jest mowa o łowieniu ryb, czy też o wkładaniu ich do puszek? Wieloznaczność zdania wynika z wieloznaczności słów wchodzących w jego skład (*can* może oznaczać czasownik *móc* lub *puszkować*, *fish* to rzeczownik *ryba* lub czasownik *łowić* *ryby*).

- problem wieloznaczności złożonych grup rzeczownikowych:

Czy grupa *small car factory* oznacza małą fabrykę samochodów, czy też fabrykę małych samochodów? Wieloznaczność frazy wynika z dwóch możliwych do przyjęcia interpretacji ośrodka (rdzenia) frazy - ośrodkiem grupy może być rzeczownik *factory* lub rzeczownik złożony *car factory*.

- problem wieloznaczności na poziomie fraz:

Czy w zdaniu *Visiting relatives can be boring* chodzi o krewnych, którzy nas odwiedzają, czy też o krewnych, których my odwiedzamy? Wieloznaczność zdania wynika z dwóch interpretacji funkcji syntaktycznej pełnionej przez słowo *visiting* - może być ono traktowane

² Jednym z niewątpliwych osiągnięć japońskiego projektu komputerów piątej generacji (ang. FGCS) było skonstruowanie w roku 1991 pierwszego modelu równoległej maszyny wnioskującej (ang. Parallel Inference Machine - PIM) [Shapiro, Warren, 1993].

jako przydawka poprzedzającego ją rzeczownika lub też jako tzw. rzeczownik odsłowny (ang. gerund).

- problem odniesień anaforycznych:

Czy ze zdania *John said to Peter that he has his hat* można wywnioskować kto ma kapelusz i kto jest jego właścicielem? Wieloznaczność wynika z trudności w określeniu, które rzeczowniki są zastępowane przez zaimki *he* i *his*.

W wielu przypadkach prawidłowe rozwiązanie wieloznaczności wymaga już nie tylko wiedzy lingwistycznej, ale również wiedzy o świecie (ang. world knowledge) i kontekstu zdania (ang. context).

1.3. Przetwarzanie języka naturalnego

W procesie automatycznego przetwarzania języka naturalnego można wyróżnić następujące zagadnienia:

- przechowywanie i manipulowanie materiałem leksykalnym (słowniki),
- analiza syntaktyczna wypowiedzi,
- analiza semantyczna wypowiedzi,
- reprezentacja wiedzy zawartej w analizowanych tekstach oraz wnioskowanie na jej elementach,
- generowanie wypowiedzi.

Zagadnienia owe bardzo rzadko występują jako odrębne i niezależne problemy do rozwiązania. Najczęściej są ze sobą ściśle powiązane, jak analiza syntaktyczna i semantyczna w gramatyce Montague [Dowty, Wall, Peters, 1981] czy w systemie SHRDLU [Winograd, 1972], oraz wzajemnie korzystają ze swoich rozwiązań, jak w przypadku zależności modułu generowania wypowiedzi od sposobu reprezentacji wiedzy. Stąd wyróżnianie ich w NLP procesie ma charakter raczej pojęciowy.

Wyżej wymienione zagadnienia zostaną omówione w kolejnych rozdziałach.

1.3.1. Słowniki

W skład każdego NLP systemu wchodzi komponent słownikowy, którego zadaniem jest przechowywanie i manipulacja materiałem leksykalnym (wyrazami, jednostkami frazeologicznymi) pojawiającym się w analizowanych tekstach. W procesie analizy tekstu wejściowego słownik rozpoznaje formy wyrazowe i zaopatruje je w informacje niezbędne

do dalszego przetwarzania. W procesie syntezy tekstu wyjściowego słownik generuje formy wyrazowe na podstawie danych dostarczonych przez inne moduły NLP systemu. Zawartość informacyjna słownika jest określana przez zastosowane w systemie metody analizy i syntezy tekstu oraz reprezentacji wiedzy³. Według [Warwick, 1987] słownik komputerowy musi opisywać dane w logiczny i spójny sposób, umożliwiać łatwe dodawanie nowych słów oraz skojarzonych z nimi informacji, wreszcie radzić sobie z olbrzymim rozmiarem przechowywanych danych.

1.3.2. Analiza syntaktyczna

Pierwszym krokiem w NLP procesie jest najczęściej analiza syntaktyczna wypowiedzi (ang. parsing). Zadaniem analizy syntaktycznej jest sprawdzenie, czy zdanie jest poprawne pod względem syntaktycznym, tzn. czy jego składnia jest zgodna z gramatyką danego języka. Podczas analizy następuje zmiana liniowego zapisu tekstu na strukturę drzewiastą, zawierającą informacje o związkach i zależnościach pomiędzy wyrazami i częściami analizowanego zdania⁴. W trakcie analizy NLP system korzysta ze słownika, który umożliwia zidentyfikowanie i sklasyfikowanie poszczególnych słów, często również występujących pomiędzy nimi niektórych relacji.

Pomimo że analiza syntaktyczna jest dopiero pierwszą fazą w NLP procesie, to już na jej etapie konieczne się staje rozwiązanie wielu problemów. Winograd [1983] charakteryzuje je następująco:

1. Wybór formalizmu (ang. choice of formalism) - w jakiej notacji, czy systemie powinna być napisana gramatyka (zwykle bowiem syntaktyczna budowa zdań definiowana jest w postaci jakiegoś formalizmu gramatycznego)?
2. Całkowitość (ang. completeness) - czy gramatyka powinna opisywać wszystkie możliwe zdania wejściowe dla danej dziedziny wiedzy, czy tylko pewien ich podzbiór? Jak reagować w przypadku zdania błędnie sformułowanego stylistycznie⁵?

³ Krańcowym przypadkiem wymagań stawianych słownikowi w NLP systemie może być słownik w systemie PAULINE, który zawiera całą lingwistyczną wiedzę [Hovy, 1988].

⁴ Podejmowano próby parsowania bez użycia komponentu syntaktycznego (ang. parsing without syntax), lecz albo były one mocno ograniczone lub też je zarzucono, jak w przypadku systemu PLANES, który został wzbogacony już jako system JETS [Tennant, 1980] o komponent syntaktyczny.

⁵ Czasami w definicji gramatyki unika się sformułowania 'niegramatycznej sekwencji' wychodząc z założenia, że niezwykle trudno jest w praktyce rozgraniczyć zdania gramatyczne od niegramatycznych (zob. [Jensen, 1988]).

3. Uniwersalność (ang. generality) - czy należy pisać oddzielne gramatyki dla odrębnych typów ludzkiej działalności językowej, czy też jedna gramatyka powinna opisywać możliwie szeroki zakres językowy?

4. Wieloznaczność (ang. ambiguity) - w jaki sposób gramatyka powinna radzić sobie z wieloznacznościami pojawiającymi się w tekstach języka naturalnego?

O złożoności powyższych problemów niech świadczy liczba opracowanych do dnia dzisiejszego formalizmów gramatycznych lub całych teorii lingwistycznych. Wymieńmy z konieczności tylko niektóre z nich (wszystkie dotyczą analizy języka angielskiego):

- analiza bez gramatyki - NLP system nie przeprowadza analizy syntaktycznej, lecz jedynie stosuje tzw. metodę uzgadniania wzorców (ang. pattern matching); metodę tą zastosowano np. w systemie ELIZA [Weizenbaum, 1966] i PARRY [Schank, Colby, 1973].
- gramatyki oparte o gramatykę bezkontekstową [Chomsky, 1956] - gramatyka transformacyjna TG [Chomsky, 1965], gramatyka Montague [1970], rozszerzona teoria standardowa EST [Chomsky, 1972], teoria GB [Chomsky, 1981], teoria X [Jackendoff, 1977];
- gramatyki funkcyjne - gramatyka systemowa Halliday'a [1967], gramatyka stratyfikacyjna [Lamb, 1966], gramatyka przypadków głębokich [Fillmore, 1968], unifikacyjna gramatyka funkcyjna [Kay, 1979].
- gramatyka ATN [Woods, 1970].
- gramatyki logiczne - gramatyka metamorficzna [Colmerauer, 1978], gramatyka DCG [Pereira, Warren, 1980].
- uogólniona gramatyka struktur frazowych GPSG [Gazdar, Klein, Pullum, Sag, 1985].
- gramatyka lewostronnie asocjacyjna [Hausser, 1989].
- gramatyka kognitywna [Langacker, 1987].

1.3.3. Analiza semantyczna

Kolejnym etapem w NLP procesie jest analiza semantyczna wypowiedzi. Zadaniem analizy semantycznej jest określenie znaczenia wypowiedzi. Najczęściej realizowane jest to poprzez transformację zdania na pewną postać formalną, która w sposób już jednoznaczny determinuje jego znaczenie. Przekształcenie to jest ściśle powiązane z zastosowaniem w procesie interpretacji semantycznej wiedzy o świecie oraz bieżącego kontekstu wypowiedzi.

W istniejących NLP systemach stosuje się bardzo różne strategie interpretacji semantycznej:

- gramatykę przypadków głębokich Fillmore'a [1968].

- gramatyki semantyczne (ang. semantic grammars); zastosowane np. w systemach SOPHIE [Brown, Burton, 1975], LIFER [Hendrix i inni, 1978], PLANES [Waltz, 1978].
- przeplot analizy syntaktycznej i semantycznej (ang. interleaved syntactic and semantic analyzer); przykładem tej techniki są systemy SHRDLU [Winograd, 1972], RUS [Bobrow, Webber, 1980].
- analizę regułową (ang. rule-by-rule semantic interpretation); wykorzystaną w gramatyce Montague [Dowty, Wall, Peters, 1981], czy w systemie DIAGRAM [Robinson, 1982].
- parsing sterowany semantyką (ang. semantically directed parsing) [Lytinen, 1986].

Postacią formalną, na którą tłumaczone są analizowane zdania to najczęściej różnego rodzaju formy logiczne ([Hobbs, Shieber, 1987], Allen [1987], [Dobryjanowicz i inni, 1991]) oraz sieci semantyczne ([Schank, Colby, 1973]⁶, [Fahlman, 1983], [Nebel, 1990], [Sowa, 1991]).

1.3.4. Reprezentacja wiedzy

Termin reprezentacja wiedzy, pomimo że stanowi podstawowe pojęcie sztucznej inteligencji (ang. artificial intelligence - AI), nadal jest przedmiotem sporów interpretacyjnych (zob. [Mildred, Shaw, Woodward, 1990]). W tym miejscu musimy zadowolić się stwierdzeniem, że reprezentacja wiedzy to pewien formalizm zapisu informacji posiadanych przez system (w szczególności NLP system) wraz ze sposobami operowania na zbiorze tych informacji. Biorąc pod uwagę stosowaną notację, algorytmy selekcji, wnioskowania i aktualizacji wiedzy można dokonać następującej klasyfikacji dostępnych metod reprezentacji wiedzy:

- reprezentacje logiczne,
- reprezentacje sieciowe,
- reprezentacje proceduralne.

Reprezentacje logiczne (np. zastosowane w systemach LUNAR [Woods, 1977], PLIDIS [Berry-Rogghe, Wultz, 1978]) oparte są na notacji i metodach wnioskowania stosowanych w logice matematycznej, w szczególności na rachunku predykatów. Reprezentacje logiczne są przykładem typowo deklaratywnego sposobu reprezentacji, wiedza jest zbiorem asercji dotyczącym modelowanej dziedziny, a wnioskowanie polega na

⁶ teoria CD Schank'a często traktowana jest jako szczególnego rodzaju sieć semantyczna (zob. [Charniak, Wilks, 1976]).

zastosowaniu jednej z klasycznych, monotonicznych⁷ metod dowodzenia twierdzeń (zob. [Kowalski, 1989]).

Reprezentacje sieciowe (np. teoria CD Schanka [Schank, Colby, 1973]) z natury swojej umożliwiają odwzorowanie niektórych struktur organizacyjnych bazy wiedzy, takich jak: dziedziczenie, generalizację, agregację, klasyfikację. W reprezentacjach opartych o sieci semantyczne wykorzystuje się metody wnioskowania zarówno formalne, jak i proceduralne.

W reprezentacji proceduralnej (np. system aktorów Hewitt'a [1973]) wiedza o danym pojęciu równoważna jest z zachowaniem się procedury reprezentującej to pojęcie. Reprezentacje proceduralne wykorzystują najczęściej różne techniki wnioskowania niemonotonicznego⁸.

Jedną z najbardziej obiecujących metod reprezentacji wiedzy jest tzw. ramowy system reprezentacji wiedzy (ang. framework). System ten łączy podejście deklaratywne i proceduralne w postaci tzw. powiązania procedur z danymi (ang. procedural attachment). Ramy zastosowano między innymi przy definiowaniu języka reprezentacji wiedzy KRL [Bobrow, Winograd, 1977].

1.3.5. Generacja wypowiedzi

Problem generacji wypowiedzi stanowi przedmiot intensywnych badań dopiero od niedawna, a proponowane obecnie rozwiązania dotyczą jedynie wybranych jego aspektów (zob. [McDonald, Bolc, 1988], [Dale, Mellish, Zock, 1990]). Pełna realizacja zadania generowania wypowiedzi w języku naturalnym wciąż nie jest możliwa z uwagi na brak odpowiednich teorii opisujących w sposób całościowy zasad rządzących funkcjonowaniem języka jako środka komunikowania się.

W procesie generowania wypowiedzi można wyróżnić następujące etapy:

- ustalenie treści - wybór faktów odpowiadających pytaniu użytkownika wraz z uwzględnieniem kontekstu wypowiedzi (modelu dialogu) i wiedzy o użytkowniku (model użytkownika),
- przygotowanie planu - dobranie kolejności przedstawiania poszczególnych faktów, wybór tematów, zawartości oraz form kolejnych zdań,
- ostateczne sformułowanie wypowiedzi - wybór słów reprezentujących poszczególne pojęcia, uzgodnienie form gramatycznych.

⁷ Monotoniczność konsekwencji logicznej polega na tym, że jeżeli do zbioru przesłanek dodamy nowy element, to wówczas wszystkie wyprowadzone dotąd wnioski pozostaną prawdziwe, a być może będzie można udowodnić jeszcze dodatkowe wnioski.

⁸ We wnioskowaniu niemonotonicznym dodanie nowego założenia może spowodować unieważnienie pewnych istniejących już twierdzeń.

1.4. Cel i zakres pracy

Już tylko pobieżne przedstawienie problematyki związanej z przetwarzaniem języka naturalnego uwidacznia olbrzymią złożoność zagadnienia, które zapewne nie będzie w najbliższym czasie w pełni rozwiązane. Toteż niniejsza praca nie stanowi próby przedstawienia kompleksowego rozwiązania problemu, choć porusza wszystkie wspomniane w poprzednich rozdziałach kwestie związane z tworzeniem NLP systemu.

Celem pracy jest określenie formalnej reprezentacji wiedzy wyrażonej w zdaniach języka naturalnego (angielskiego) w bazie wiedzy NLP systemu. System ten powinien posiadać zdolność do dokonania automatycznej dekompozycji zdania na takie elementy bazy wiedzy, które zapewniają przeprowadzenie analizy syntaktycznej, integrację wiedzy oraz umożliwiają przeprowadzenie analizy semantycznej i udzielenie odpowiedzi (poprzez proces poszukiwania lub wnioskowania) na zadane pytanie. Zarówno pytanie, jak i odpowiedź powinny mieć formę zdań języka angielskiego. Ze względu na złożoność analizowanych zagadnień przyjęto, że komunikacja z NLP systemem odbywa się wyłącznie w formie pojedynczych zdań języka angielskiego.

Za podstawę formalnej reprezentacji wiedzy przyjęto model zaproponowany i rozwijany przez J. Kazimierczaka [1985, 1987, 1989, 1990, 1991, 1993]. W pracy przedstawiono istotne modyfikacje modelu, a co za tym idzie również rozszerzenie występującego w modelu formalizmu wyrażeń symbolicznych.

Zaproponowano oryginalną metodę dekompozycji zdania języka angielskiego, spełniającą funkcję analizy syntaktycznej, na cząstki wiedzy. Dekompozycja realizowana jest przez niedeterministyczny automat skończony ze stosami i skojarzeniowym dostępem do pamięci [Unold, 1992c].

Przedstawiono pewien sposób reprezentacji angielskiej grupy werbalnej osobowej, który minimalizuje zapamiętywaną informację o składni grupy do niezbędnego minimum oraz ułatwia dalszą analizę semantyczną grupy [Unold, 1992a, 1992b].

W pracy opisano również koncepcję analizy semantycznej zdekomponowanego zdania wykorzystującej zmodyfikowaną gramatykę przypadków głębokich Halliday'a.

Przedstawiono także możliwość przeprowadzenia wnioskowania proceduralnego w proponowanym modelu reprezentacji wiedzy, zakończonego generowaniem odpowiedzi w języku angielskim.

Praca składa się z 5-ciu rozdziałów. W rozdziale 1 przedstawiono wprowadzenie do tematu, cel i zakres pracy. Rozdział 2 prezentuje formalny model dekompozycji zdania na elementy bazy wiedzy. W podrozdziałach rozdziału 2 wprowadzono pojęcie komponentu strukturalno-semantycznego, przedstawiono model formalny pewnego automatu ze stosami nazywanego automatem *PDAMS* wraz z jego realizacją programową. Rozdział 3 opisuje

formalny system reprezentacji wiedzy wyrażonej w zdaniu języka naturalnego. W kolejnych podrozdziałach rozdziału 3 przedstawiono formalną reprezentację komponentu o strukturze syntaktycznej grupy werbalnej osobowej, transformację wiedzy wyrażonej w zdaniu języka angielskiego na postać symboliczną, syntezę wiedzy w bazie wiedzy, wreszcie proces reprodukcji wiedzy. W rozdziale 4 zaprezentowano możliwości przeprowadzenia wnioskowania proceduralnego na bazie wiedzy, przy wykorzystaniu zmodyfikowanego zbioru kategorii semantycznych i mechanizmu klasyfikacji gramatyki systemowej. Rozdział 5 zawiera podsumowanie.

Ponadto w załącznikach do pracy zamieszczono tabulogram programu, będącego realizacją programową automatu *PDAMS* oraz algorytmy analizy i syntezy gramatycznej komponentu o strukturze syntaktycznej grupy werbalnej osobowej.

2. DEKOMPOZYCJA ZDANIA NA ELEMENTY BAZY WIEDZY

Przyjęty w pracy model reprezentacji wiedzy bazuje na koncepcji J. Kazimierczaka. Baza wiedzy (ang. Knowledge Base - KB) NLP systemu opartego na tej koncepcji reprezentowana jest przez odpowiednie wyrażenia symboliczne i słowniki (zob. [Kazimierczak, 1990]). Wiedza nabywana ze zdań języka naturalnego jest nakładana na wiedzę już istniejącą (synteza wiedzy). Proces nabywania i integracji wiedzy zorganizowany jest w taki sposób, że ten sam element wiedzy (zawarty w odpowiadającej mu jednostce językowej) i pojawiający się w wielu przychodzących na wejście systemu zdaniach, reprezentowany jest tylko przez jeden element symboliczny w KB systemu.

Odwzorowanie tekstu języka naturalnego (angielskiego) reprezentującego nabywaną wiedzę na postać symboliczną wymaga rozróżnienia w analizowanym tekście poszczególnych jednostek językowych, tzn. słów, grup słów, zdań. Ograniczenie wejściowego tekstu do formy pojedynczych zdań języka angielskiego ogranicza zadanie dekompozycji wypowiedzi do zadania dekompozycji zdania na grupy słów oraz na poszczególne słowa.

2.1. Dekompozycja zdania na komponenty strukturalno-semantyczne

Aby zwiększyć integralność wiedzy w KB i efektywność algorytmu przeszukiwania KB przyjęto następujące założenia określające strukturę grup słów, na które należy podzielić zdanie:

- w zdaniu należy wyróżnić grupy słów, które stanowią niepodzielne już jednostki semantyczne, każda o charakterystycznej strukturze składniowej. Używając pojęcia syntagma [Bünting, 1985] opisującego ogólnie grupę wyrazów występujących łącznie, można powiedzieć, że zdanie musi zostać zdekomponowane na elementarne syntagmy semantyczne.

Elementarna syntagma semantyczna nie jest tożsama z grupą słów tworzącą klasę gramatyczną. Na przykład grupa słów *the pictures in my room* składniowo tworzy syntaktyczną frazę rzeczownikową (nominalną) [Smólska, 1976], lecz nie jest atomem semantycznym. W grupie tej można wyróżnić dwie elementarne syntagmy semantyczne: *the pictures* o strukturze syntaktycznej grupy nominalnej oraz grupę *in my room* o strukturze syntaktycznej grupy przyimkowej.

● w zdaniu należy wyróżnić wszystkie możliwe elementarne syntagmy semantyczne, również w tym wypadku, gdy są wbudowane w strukturę innych elementarnych syntagm semantycznych. Przypadek ten zachodzi, gdy syntaktyczną rolę przydawki w syntagmie spełnia inna syntagma semantyczna⁹.

Przykładowo, w elementarnej syntagmie o strukturze syntaktycznej grupy nominalnej *a very long rather boring historical novel* (zob. [Smólska, 1976] na str. 122) występują trzy przydawki, każda o strukturze grupy przymiotnikowej: *very long*, *rather boring* i *historical*. Co więcej, dwie pierwsze z wymienionych przydawek mają własne przydawki¹⁰ o strukturze grupy przysłówkowej: *very* i *rather*.

Warto przy okazji zauważyć, że przymiotnik *long* pełni w całej grupie nominalnej jedynie rolę syntaktyczną, dopełniającą znaczenia ośrodka (rdzenia) grupy - rzeczownika *novel*. Natomiast ten sam przymiotnik *long* w kontekście grupy przymiotnikowej *very long* stanowi już jej rdzeń semantyczny.

Tak zdefiniowaną grupę słów będziemy nazywali komponentem strukturalno-semantycznym (komponentem). Odtąd, pod pojęciem dekompozycji zdania na grupy słów będziemy rozumieli wyodrębnienie wszystkich komponentów strukturalno-semantycznych w zdaniu.

2.2 Dekompozycja zdania na grupy słów

Aby dekompozycja zdania na komponenty strukturalno-semantyczne była możliwa, baza wiedzy NLP systemu musi posiadać informację o kategoriach leksykalnych poszczególnych słów tworzących zdanie. Informacja ta potrzebna jest nie tylko do podziału zdania na komponenty, lecz zgodnie z założeniami metody J. Kazimierczaka również do sprawdzenia poprawności zdania pod względem syntaktycznym. Podczas syntezy początkowej KB (tzn. gdy jest ona pusta lub niepełna) użytkownik systemu (inżynier wiedzy) musi wraz ze zdaniem podać dodatkową informację w formie ciągu symboli oznaczających kategorie leksykalne poszczególnych słów zdania (jak w pracy [Kazimierczak, 1989]). Przyjmuje się następujący kod kategorii leksykalnych słów języka angielskiego:

w₀ - znacznik końca zdania (kropka),

w₁ - rzeczownik (ang. noun),

w₂ - zaimek (ang. pronoun),

w₃ - określnik (ang. determiner),

⁹ Zjawisko to charakteryzuje angielski termin rankshift.

¹⁰ Są to tzw. przydawki wtórne (ang. submodifiers).

- w₄ - przymiotnik (ang. adjective),
- w₅ - czasownik (ang. verb),
- w₆ - przysłówek (ang. adverb),
- w₇ - przyimek (ang. preposition)
- w₈ - spójnik (ang. conjunction),
- w₉ - liczebnik (ang. number).

Podział zdania na komponenty strukturalno-semantyczne sprowadza się do problemu podziału ciągu symboli reprezentujących kategorie leksykalne słów w zdaniu na podciągi.

Każdy ciąg symboli oznaczających kategorie leksykalne poszczególnych wyrazów analizowanego zdania (nazywany dalej ciągiem (w)) można traktować jako słowo pewnego języka formalnego. Okazuje się, że dopiero gramatyki kontekstowe są wystarczająco pojemne, aby mogły posłużyć do opisu języka naturalnego [Chomsky, 1965]. Jednak najczęściej podczas procesu dekompozycji nie występuje konieczność wykorzystania wszystkich właściwości i cech języka naturalnego - angielskiego. Na niepełne wykorzystanie mocy gramatyk kontekstowych składa się między innymi praktyczne ograniczenie liczby wyrazów w analizowanych zdaniach do kilkunastu¹¹. Bardzo długie zdania, nawet poprawnie zbudowane, są najczęściej niezrozumiałe i rzadko używane. Można przyjąć, że ciąg (w) jest słowem generowanym przez gramatykę bezkontekstową. Problem dekompozycji zdania języka angielskiego zastępowany jest więc przez problem znalezienia specyficznego znacznika frazowego pewnego słowa [Blikle, 1971], [Kowalski, Mostowski, 1979]. Jak wiadomo, język generowany przez gramatykę bezkontekstową jest akceptowany przez automat ze stosem [Chomsky, 1962].

2.3. Model formalny automatu dekomponującego ciąg (w) na podciągi (w)_i

Konstrukcja i zasada działania automatu dokonującego podziału ciągu (w) na podciągi (w)_i powinny uwzględniać następujące własności dekomponowanego zdania języka angielskiego:

- w języku angielskim dwa zdania dekomponujące się na różne komponenty strukturalno-semantyczne mogą mieć te same początkowe podciągi odpowiadających im ciągów (w) i nie można z góry określić długości tych podciągów.

Rozważmy przykładowe dwa zdania: *The train arrives early* i *The train will not arrive early*. Zdaniom tym odpowiadają odpowiednio następujące ciągi (w), definiujące kategorie

¹¹ Podczas niemal rocznej pracy systemu CRITQUE setki użytkowników wprowadziło do systemu 66.301 zdań. Średnia liczba wyrazów w zdaniu wyniosła trzynaście [Jensen, 1988].

leksykalne słów tworzących zdania: $w_3w_1w_5w_6w_0$ i $w_3w_1w_5w_6w_5w_6w_0$. Pierwsze cztery słowa obu zdań mają te same kategorie leksykalne (odpowiada im podciąg $w_3w_1w_5w_6$). Jednak w zdaniu pierwszym przysłówek *early* (czwarte słowo zdania) tworzy odrębny komponent o kategorii grupy przysłówkowej, natomiast w zdaniu drugim przysłówek *not* (czwarte słowo zdania) jest jedynie elementem grupy werbalnej *will not arrive*.

- dwa elementarne komponenty strukturalno-semantyczne o różnych kategoriach składniowych mogą mieć te same początkowe podciągi odpowiadających im ciągów (w) i nie można z góry określić długości tych podciągów.

Przykładowo, komponentowi *the student* o kategorii grupy nominalnej odpowiada ciąg (w) w_3w_1 , natomiast komponentowi o kategorii grupy przysłówkowej *an hour ago* ciąg $w_3w_1w_6$. Oba komponenty mają jednakowy początkowy podciąg (w)₁ w_3w_1 .

- pojedynczy wyraz może mieć więcej niż jedną interpretację leksykalną.

Słowo *well* może oznaczać rzeczownik, czasownik, przymiotnik lub przysłówek [Longman..., 1987].

Wyżej wymienione własności zdań języka angielskiego prowadzą do wniosku, że w automacie dokonującym podziału zdania funkcja przejść musi zostać zastąpiona relacją przejść. Stąd, aby przeprowadzić deterministyczną symulację niedeterministycznego automatu należy dla każdego zbioru alternatywnych przejść automatu określić deterministyczną funkcję wyboru przejścia. W związku z możliwością wybrania niewłaściwego przejścia automat musi mieć możliwość skasowania ruchu lub sekwencji ruchów, które nie prowadzą do akceptacji (i podziału) wejściowego słowa.

Warunek minimalnej liczby stanów automatu prowadzi do wniosku, że w strukturze automatu należy wyróżnić pewne podautomaty stanowe¹² akceptujące poszczególne komponenty strukturalno-semantyczne (tzn. każdy podautomat stanowy akceptuje pewien zbiór podciągów (w)_i i dodatkowo zbiory te są rozłączne).

Organizacja dostępu do stosu jest w większości modeli automatowych typu FILO [Revesz, 1983]. Aby zwiększyć efektywność dostępu automatu do pamięci w jego definicji wykorzystano koncepcję skojarzeniowego dostępu do pamięci (ang. associative memory access), w której zastosowano pojęcie markera [Weicker, 1974]. Marker (znacznik) jest wyróżnionym elementem alfabetu taśmy automatu. Głowica automatu obok ruchów typu klatka-w-lewo i klatka-w-prawo może wykonywać skoki do najbliższej klatki z markerem.

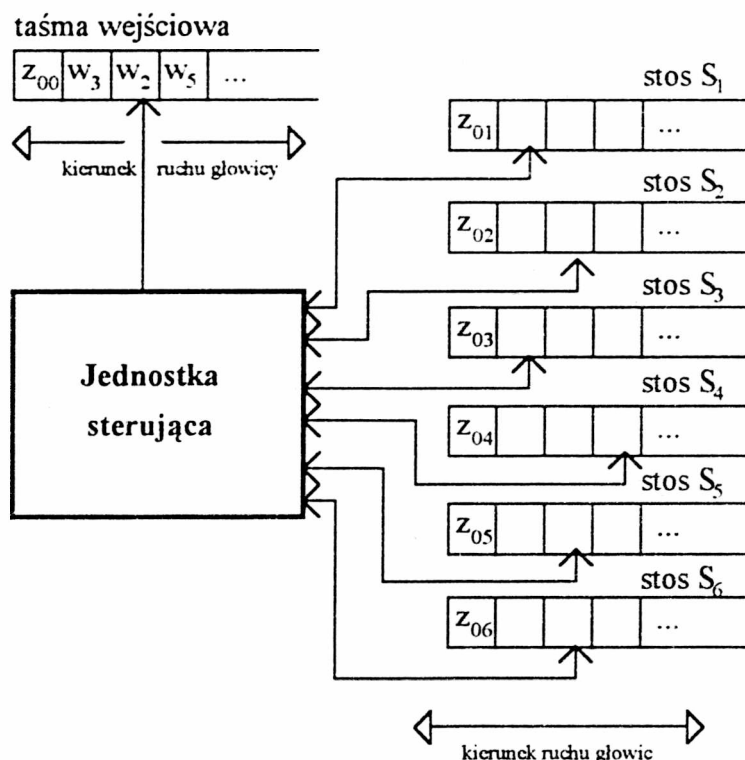
Automat dekomponujący ciąg (w) na podciągi (w)_i, a w konsekwencji dzielący zdanie języka angielskiego na komponenty strukturalno-semantyczne będziemy nazywali

¹²Podautomat stanowy powstaje w wyniku wydzielenia z automatu podzbioru stanów wewnętrznych [Mikołajczak, 1985].

automatem ze stosami i skojarzeniowym dostępem do pamięci i oznaczali symbolem *PDAMS* (ang. Pushdown Automata with Associative Memory Access).

2.3.1. Struktura automatu

Na rys. 1 przedstawiono strukturę automatu *PDAMS*, realizującego dekompozycję ciągu symboli (w) na podciągi $(w)_i$. Automat składa się z jednostki sterującej, taśmy wejściowej, na której umieszczone jest słowo wejściowe oraz sześciu taśm roboczych. Organizacja dostępu do taśmy roboczej jest taka, że jeśli jakiś symbol ma być umieszczony na niej, to zajmuje on pierwszą klatkę na prawo od klatki, w której ostatnio zapisano symbol. Przy odczytywaniu symbolu z taśmy roboczej pobierany jest symbol, który został zapisany jako ostatni. Taśma robocza ma więc strukturę stosu, a automat *PDAMS* stanowi pewien typ automatu ze stosami. Zarówno taśma wejściowa jak i stosy są prawostronnie nieskończone. Z taśmą wejściową współpracuje głowica czytająca, ze stosami głowice czytająco-zapisujące. Każda taśma składa się z ciągu klatek. W klatce może być umieszczony jeden symbol ze skończonego alfabetu taśmy. W lewej skrajnej klatce umieszczony jest symbol oznaczający lewy koniec taśmy. Głowica taśmy wejściowej i głowice stosów mogą przesuwać się zarówno w lewo, jak i w prawo.



Rys. 1. Ogólna struktura automatu *PDAMS*

Przeznaczenie poszczególnych stosów automatu *PDAMS* można scharakteryzować następująco, poczynając od stosu S_3 .

Na stosie S_3 zapamiętywane są kolejno czytane symbole z taśmy wejściowej, które tworzyć będą podciąg $(w)_i$ ciągu (w) . Podciąg ten rozpoczynają zapisane na stosie wyróżniane z alfabetu stosowego dwa symbole. Zadaniem pierwszego z nich, należącego do zbioru L i nazywanego l-markerem, jest m.in. określenie kategorii syntaktycznej wyodrębnianego podciagu. Drugi z wyróżnianych symboli, nazywany t-markerem i należący do zbioru T , wskazuje na początek podciagu. Przykładowa zawartość stosu S_3 podczas pracy automatu *PDAMS* może być następująca: $z_0l_1t_1w_1$. Symbol z_0 oznacza lewy koniec taśmy S_3 , marker l_1 stanowi, że obecnie tworzony podciąg należy do kategorii syntaktycznej grupy nominalnej, marker t_1 poprzedza pierwszy element tworzonego podciagu, którym jest symbol w_1 (reprezentujący rzeczownik).

Stosy S_1 , S_2 i S_5 służą do ewentualnego zapamiętania elementów aktualnej konfiguracji automatu (definicja konfiguracji zostanie podana w następnym rozdziale). Na stosie S_1 zapamiętywany jest symbol aktualnego stanu automatu. Liczba przeczytanych symboli z taśmy wejściowej składowana jest na stosie S_2 . Na stosie S_5 zapisywana jest zawartość stosu S_3 poprzedzona wyróżnionym elementem nazywanym p-markerem. Zadaniem p-markera zapisanego na stosie S_5 jest oddzielanie kolejnych zapisów zawartości stosu S_3 .

Na stos S_4 wkładany jest symbol tego stanu automatu, który stanie się jego stanem aktualnym po powrocie sterowania z jednego z podautomatów.

Na stosie S_6 zapisywane są kolejno tworzone podciagi $(w)_i$ ciągu (w) . Każdy podciąg $(w)_i$, który składa się z symboli w_j , rozpoczyna na stosie S_6 symbol t_1 , natomiast kończy symbol t_2 ze zbioru T . Przed symbolem t_1 wskazującym na początek podciagu zapisywany jest l-marker definiujący kategorię syntaktyczną podciagu. Przykładowo, chwilowa zawartość stosu S_6 może być następująca: $z_0l_1t_1w_1t_2l_2t_1w_5w_5t_2$. Symbol z_0 oznacza lewy koniec taśmy S_6 , marker l_1 określa, że podciąg ograniczony przez najbliższe symbole t_1 i t_2 , czyli podciąg składający się z pojedynczego elementu w_1 , należy do kategorii syntaktycznej grupy nominalnej, kolejny marker l_2 definiuje podciąg w_5w_5 jako grupę werbalną. Mogący pojawić się na stosie S_6 p-marker wskazuje na miejsce, od którego definicje stworzonych podciągów mogą jeszcze ulec zmianie w trakcie działania automatu *PDAMS*.

W pracy przy zapisie markerów t_1 i t_2 ze zbioru T stosować będziemy również odpowiednio symbol nawiasu lewego "(" oraz symbol nawiasu prawego ")". W takiej notacji przedstawiona wyżej przykładowa zawartość stosu S_6 będzie postaci: $z_0l_1(w_1)l_2(w_5w_5)$.

Zakładamy, że automat pracuje według następujących ogólnych zasad:

- 1) automat działa w czasie dyskretnym,
- 2) w każdym okresie aktywności automat może wykonać równocześnie następujące czynności:
 - zapisać nowe symboli na wybranych stosach,
 - przesunąć wybrane głowice niezależnie w lewo lub w prawo o skończoną liczbę klatek,
 - zmienić stan.
- 3) działanie automatu zdeterminowane jest przez pewną skończoną liczbę rozkazów zwanych instrukcjami (definicja instrukcji automatu zostanie podana w dalszej części pracy).

2.3.2. Definicja formalna automatu

Formalnie, pod pojęciem automatu *PDAMS* będziemy rozumieli system

$$PDAMS = (Q, Z, F, \delta, z_0, q_0, q_f, L, T, p), \text{ gdzie}$$

Q jest skończonym zbiorem stanów wewnętrznych automatu ($q_i \in Q$),

$q_0 \in Q$ jest wyróżnionym stanem zwanym stanem początkowym,

$q_f \in Q$ jest wyróżnionym stanem zwanym stanem końcowym,

$F \subset Q$ jest wyróżnionym podzbiorem zbioru Q zwanym zbiorem stanów końcowych podautomatów,

L jest skończonym zbiorem tzw. l-markerów ($L = \{l_1, l_2, l_3, l_4, l_5, l_6\}$),

T jest skończonym zbiorem tzw. t-markerów ($T = \{t_1, t_2\}$),

p jest elementem zwanym p-markerem,

$Z = W \cup Z_1 \cup Z_2 \cup Z_3 \cup Z_4 \cup Z_5 \cup Z_6$, gdzie

W jest alfabetem taśmy wejściowej ($W = \{w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9\}$),

Z_1 jest alfabetem stosu S_1 ($Z_1 = Q$),

Z_2 jest alfabetem stosu S_2 (Z_2 jest zbiorem liczb naturalnych),

Z_3 jest alfabetem stosu S_3 ($Z_3 = W \cup T \cup L$),

Z_4 jest alfabetem stosu S_4 ($Z_4 = Q$),

Z_5 jest alfabetem stosu S_5 ($Z_5 = W \cup T \cup L \cup p$),

Z_6 jest alfabetem stosu S_6 ($Z_6 = W \cup T \cup L \cup p$),

$z_0 = (z_{00}, z_{01}, z_{02}, z_{03}, z_{04}, z_{05}, z_{06}) \in W \times Z_1 \times Z_2 \times Z_3 \times Z_4 \times Z_5 \times Z_6$ jest siódmką symboli początkowych odpowiednio taśmy wejściowej i stosów.

Niech $M_i \in (d \times \{m_l, m_h, m_r\})$ dla $i = 0, 1, \dots, 6$ oznacza przesunięcie i -tej głowicy takie, że m_h oznacza bezruch głowicy, m_l - przesunięcie głowicy w lewo o d -klatek, m_r - przesunięcie głowicy w prawo o d -klatek.

δ jest relacją przejść odwzorowującą

$$\Phi \rightarrow 2^{\Phi \times \Theta}$$

gdzie

$$\Phi = Q \times W \times Z_1 \times Z_2 \times Z_3 \times Z_4 \times Z_5 \times Z_6,$$

$$\Theta = M_0 \times M_1 \times M_2 \times M_3 \times M_4 \times M_5 \times M_6.$$

Definicję relacji przejść δ rozumiemy w ten sposób, że dla układu 8-elementowego, składającego się z aktualnego stanu automatu, symbolu wejściowego oraz sześciu symboli stosowych, relacja przejść określa zbiór, którego każdy element to nowy stan automatu i siedem par, z których każda określa nowy symbol taśmy wejściowej lub stosów oraz kierunek i długość ruchu głowicy.

Pracę automatu *PDAMS* można formalnie opisać za pomocą pojęcia konfiguracji. Konfiguracją automatu *PDAMS* nazywamy 7-elementowy układ

$$K = (\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6), \text{ gdzie}$$

α_0 jest trójką (q_a, x, y) taką, że q_a jest aktualnym stanem automatu, x jest słowem zapisanym na taśmie wejściowej licząc od początku taśmy do miejsca położenia głowicy, y jest słowem zapisanym na prawo od głowicy, natomiast α_i ($i = 1, \dots, 6$) jest dwójką (q_a, v) taką, że v jest słowem zapisanym na i -tym stosie (na lewo od głowicy stosu).

Jeśli automat przechodzi w jednym kroku od konfiguracji K_t do konfiguracji K_{t+1} , gdzie t oznacza chwilę czasu, to oznaczamy ten ruch za pomocą relacji binarnej $K_t \Rightarrow K_{t+1}$. Zapis $K_t \Rightarrow^* K_{t+j}$ oznacza ciąg konfiguracji taki, że $K_t \Rightarrow K_{t+1} \Rightarrow \dots \Rightarrow K_{t+j}$.

Automat *PDAMS* akceptuje (dzieli) słowo wejściowe $w_1 w_2 \dots w_n$ (ciąg (w)) zapisane na taśmie wejściowej, jeżeli istnieje $K_0 \Rightarrow^* K_f$, gdzie

$$K_0 = ((q_0, z_0 w_1, w_2 w_3 \dots w_n), (q_0, z_0), \dots, (q_0, z_0)),$$

$$K_f = (\alpha_0, \alpha_1, \dots, \alpha_6) \text{ oraz dla pewnego } i \ (0 \leq i \leq 6) \ q_f \text{ występuje w } \alpha_i.$$

Konfigurację K_0 nazywamy konfiguracją początkową automatu *PDAMS*, konfigurację K_f konfiguracją końcową. Automat *PDAMS* akceptuje słowo wejściowe (a w konsekwencji dzieli na pod słowa) w momencie osiągnięcia stanu końcowego q_f automatu.

2.3.3. Instrukcje automatu

Instrukcje są regułami przejścia, które pozwalają automatowi na przechodzenie od jednej konfiguracji do drugiej.

Niech O, N, P, M, U oznaczają słowa nad alfabetem taśmy wejściowej, tzn. $O, N, P, M, U \in W^*$,

A słowo nad alfabetem stosu S_1 ($A \in Q^*$),

B słowo nad alfabetem stosu S_2 ($B \in Z_2^*$),

C i J słowa nad alfabetem stosu S_3 ($C, J \in (W \cup T \cup L)^*$),
D słowo nad alfabetem stosu S_4 ($D \in Q^*$),
E słowo nad alfabetem stosu S_5 ($E \in (W \cup T \cup L \cup p)^*$),
G słowo nad alfabetem stosu S_6 ($G \in (W \cup T \cup L \cup p)^*$).

Niech K^* oznacza zbiór wszystkich możliwych konfiguracji automatu *PDAMS*

$$K^* = \{K \mid K_0 \Rightarrow^* K \text{ i } K = ((q_0, z_{00}w_{j1}, w_{j2}O), (q_0, z_{01}), \dots, (q_0, z_{06})), \\ K_0, K \in Q \times z_{00}W^* \times W^* \times z_{00}Z_1^* \times z_{02}Z_2^* \times \dots \times z_{06}Z_6^*\}$$

Każda instrukcja automatu *PDAMS* jest jednego z następujących siedmiu typów:

1) instrukcja typu i_1 :

wykonanie instrukcji i_1 powoduje wstawienie na stos S_3 symbolu czytanego przez głowicę taśmy wejściowej, a następnie przesunięcie głowicy taśmy wejściowej o jedną klatkę w prawo. Formalny zapis instrukcji i_1 ma postać:

$$(\exists K_1, K_2 \in K^*)[\text{jeżeli } (K_1 \Rightarrow K_2) \text{ to} \\ (K_1 = ((q_i, z_{00}Nw_{jk}, w_{jk+1}P), (q_i, z_{01}A), (q_i, z_{02}B), (q_i, z_{03}C), (q_i, z_{04}D), (q_i, z_{05}E), \\ (q_i, z_{06}G))), \\ K_2 = ((q_i, z_{00}Nw_{jk}w_{jk+1}, P), (q_i, z_{01}A), (q_i, z_{02}B), (q_i, z_{03}Cw_{jk}), (q_i, z_{04}D), (q_i, z_{05}E), \\ (q_i, z_{06}G)))]$$

2) instrukcja typu i_2 :

wykonanie instrukcji i_2 powoduje przepisanie ze stosu S_6 grupy symboli ze zbioru W , licząc od szczytu stosu do pierwszego l-markera oraz ciągu następujących po sobie symboli ze zbioru L , na stos S_3 . Formalny zapis instrukcji i_2 ma postać:

$$(\exists K_1, K_2 \in K^*)[\text{jeżeli } (K_1 \Rightarrow K_2) \text{ to} \\ (K_1 = ((q_i, z_{00}Nw_{jk}, w_{jk+1}P), (q_i, z_{01}A), (q_i, z_{02}B), (q_i, z_{03}C), (q_i, z_{04}D), (q_i, z_{05}E), \\ (q_i, z_{06}GRSK))), \\ K_2 = ((q_i, z_{00}Nw_{jk}, w_{jk+1}P), (q_i, z_{01}A), (q_i, z_{02}B), (q_i, z_{03}CSX), (q_i, z_{04}D), (q_i, z_{05}E), \\ (q_i, z_{06}GR)), R, X \in (W \cup T \cup p)^*, S \in L^*)]$$

3) instrukcja typu i_3 :

wykonanie instrukcji i_3 powoduje zapamiętanie elementów aktualnej konfiguracji automatu w pamięci stosowej:

- na szczyt stosu S_1 wstawiany jest symbol aktualnego stanu q_i ,
- na szczyt stosu S_2 wstawiana jest liczba h-przeczytanych symboli z taśmy wejściowej,
- na szczyt stosu S_5 zapisywany jest p-marker, a następnie zawartość stosu S_3 ,

- na szczyt stosu S_6 wstawiany jest p-marker.

Formalny zapis instrukcji i_3 ma postać:

$(\exists K_1, K_2 \in K^*)[\text{jeżeli } (K_1 \Rightarrow K_2) \text{ to}$

$(K_1 = ((q_i, z_{00}Nw_{jk}, w_{jk+1}P), (q_i, z_{01}A), (q_i, z_{02}B), (q_i, z_{03}C), (q_i, z_{04}D), (q_i, z_{05}E), (q_i, z_{06}G)),$

$K_2 = ((q_i, z_{00}Nw_{jk}, w_{jk+1}P), (q_i, z_{01}Aq_i), (q_i, z_{02}Bh), (q_i, z_{03}C), (q_i, z_{04}D), (q_i, z_{05}EpC), (q_i, z_{06}Gp)), h \text{ jest liczbą nat.})]$

4) instrukcja typu i_4 :

wykonanie instrukcji i_4 powoduje powrót automatu do konfiguracji, elementy której odczytywane są z pamięci stosowej (konfigurację tą oznaczamy symbolem K_a):

- ze szczytu stosu S_1 pobierany jest symbol stanu q_j , stan q_j staje się stanem aktualnym automatu,

- na podstawie zawartości stosu S_5 , licząc od szczytu stosu do pierwszego p-markera, zostaje odtworzona zawartość stosu S_3 automatu w konfiguracji K_a ,

- ze szczytu stosu S_2 pobierana jest liczba h -przeczytanych symboli taśmy wejściowej w chwili osiągnięcia konfiguracji K_a , głowica taśmy wejściowej przesuwana nad h -ty symbol taśmy.

- ze stosu S_6 usuwana jest grupa symboli, licząc od szczytu stosu do pierwszego p-markera.

Formalny zapis instrukcji i_4 ma postać:

$(\exists K_1, K_2 \in K^*)[\text{jeżeli } (K_1 \Rightarrow K_2) \text{ to}$

$(K_1 = ((q_i, z_{00}Nw_{jk}, w_{jk+1}P), (q_i, z_{01}Aq_j), (q_i, z_{02}Bh), (q_i, z_{03}C), (q_i, z_{04}D), (q_i, z_{05}EpJ), (q_i, z_{06}GpY)),$

$K_2 = ((q_j, z_{00}Mw_{jh}, w_{jh+1}U), (q_j, z_{01}A), (q_j, z_{02}B), (q_j, z_{03}J), (q_j, z_{04}D), (q_j, z_{05}E), (q_j, z_{06}G)), h \text{ jest liczbą nat., } Y \in (W \cup L \cup T)^*)]$

5) instrukcja typu i_5 (q_k i l_t):

wykonanie instrukcji i_5 powoduje wstawienie na szczyt stosu S_4 symbolu q_k , natomiast na szczyt stosu S_3 markerów l_t i t_1 . Formalny zapis instrukcji i_5 ma postać:

$(\exists K_1, K_2 \in K^*)[\text{jeżeli } (K_1 \Rightarrow K_2) \text{ to}$

$(K_1 = ((q_i, z_{00}Nw_{jk}, w_{jk+1}P), (q_i, z_{01}A), (q_i, z_{02}B), (q_i, z_{03}C), (q_i, z_{04}D), (q_i, z_{05}E), (q_i, z_{06}G)),$

$K_2 = ((q_i, z_{00}Nw_{jk}, w_{jk+1}P), (q_i, z_{01}A), (q_i, z_{02}B), (q_i, z_{03}Cl_t t_1), (q_i, z_{04}Dq_k), (q_i, z_{05}E), (q_i, z_{06}G)))]$

6) instrukcja typu i_6 :

wykonanie instrukcji i_6 powoduje przejście sterowania automatu do stanu pobranego ze szczytu stosu S_4 . Instrukcja oznacza powrót sterowania z podautomatu. Formalny zapis instrukcji i_6 ma postać:

$$(\exists K_1, K_2 \in K^*)[\text{jeżeli } (K_1 \Rightarrow K_2) \text{ to} \\ (K_1 = ((q_i, z_{00}Nw_{jk}, w_{jk+1}P), (q_i, z_{01}A), (q_i, z_{02}B), (q_i, z_{03}C), (q_i, z_{04}Dq_k), \\ (q_i, z_{05}E), (q_i, z_{06}G))), \\ K_2 = ((q_k, z_{00}Nw_{jk}, w_{jk+1}P), (q_k, z_{01}A), (q_k, z_{02}B), (q_k, z_{03}C), (q_k, z_{04}D), (q_k, z_{05}E), \\ (q_k, z_{06}G)))]$$

7) instrukcja typu i_7 :

wykonanie instrukcji i_7 powoduje wstawienie na szczyt stosu S_6 zawartości stosu S_3 , a następnie markera t_2 . Stos S_3 jest opróżniany. Formalny zapis instrukcji i_7 ma postać:

$$(\exists K_1, K_2 \in K^*)[\text{jeżeli } (K_1 \Rightarrow K_2) \text{ to} \\ (K_1 = ((q_i, z_{00}Nw_{jk}, w_{jk+1}P), (q_i, z_{01}A), (q_i, z_{02}B), (q_i, z_{03}C), (q_i, z_{04}D), (q_i, z_{05}E), \\ (q_i, z_{06}G))), \\ K_2 = ((q_i, z_{00}Nw_{jk}, w_{jk+1}P), (q_i, z_{01}A), (q_i, z_{02}B), (q_i, z_{03}), (q_i, z_{04}D), (q_i, z_{05}E), \\ (q_i, z_{06}GCt_2)))]$$

2.3.4. Algorytm działania automatu

Na początku działania automat *PDAMS* znajduje się w konfiguracji początkowej K_0 . Na taśmie wejściowej zapisana jest sekwencja $w_1w_2...w_n$ (ciąg (w)). Symbolem q_a oznaczać będziemy aktualny stan automatu. Zasadę działania automatu można opisać w następujących krokach:

krok 1) jeżeli aktualny stan automatu $q_a = q_f$, to zakończ działanie automatu z wynikiem pozytywnym.

Interpretacja:

jeżeli automat osiągnął stan końcowy q_f , to kończy swoje działanie. Na stosie S_6 zapisany jest ciąg (w) podzielony na podciągi $(w)_i$.

krok 2) jeżeli $q_a \in F$, to wykonaj instrukcję i_6 i przejdź do kroku 1.

Interpretacja:

jeżeli automat znajduje się w stanie końcowym jednego z podautomatów to przechodzi do stanu wskazywanego przez wierzchołek S_4 .

krok 3) $K(q_a) = \{K \mid K^a \Rightarrow K \text{ i}$

$K^a = ((q_a, z_{00}N, P), (q_a, z_{01}A), (q_a, z_{02}B), (q_a, z_{03}C), (q_a, z_{04}D), (q_a, z_{05}E), (q_a, z_{06}G)),$

$K = ((q_x, z_{00}N', P'), (q_x, z_{01}A'), (q_x, z_{02}B'), (q_x, z_{03}C'), (q_x, z_{04}D'), (q_x, z_{05}E'), (q_x, z_{06}G')) \text{ i } K^a, K \in K^*\}$

Interpretacja:

zbiór $K(q_a)$ jest zbiorem wszystkich możliwych alternatywnych ruchów (następnych konfiguracji) automatu znajdującego się w stanie q_a .

krok 4) jeżeli $\|K(q_a)\| \geq 1$, to

krok 4a) wybierz następną konfigurację automatu $K_{nas} \in K(q_a)$ i postaw $K(q_a) = K(q_a) \setminus K_{nas}$.

Interpretacja:

jeżeli automat ma w stanie q_a więcej niż jeden możliwy ruch, to należy wybrać następny ruch ze zbioru możliwych (według zadanej gramatyki), a zbiór ruchów możliwych w stanie q_a zmniejszyć o wybrany ruch.

krok 4b) jeżeli $\|K(q_a)\| > 0$, to wykonaj instrukcję i_3 .

Interpretacja:

jeżeli po wybraniu ruchu ze zbioru ruchów alternatywnych $K(q_a)$ zbiór ten jest niepusty, to należy elementy aktualnej konfiguracji zapamiętać w pamięci stosowej.

krok 4c) wykonaj ruch $K^a \Rightarrow K_{nas}$ i przejdź do kroku 1.

krok 5) jeżeli stos S_1 jest pusty, to zakończ działanie automatu z wynikiem negatywnym.

Interpretacja:

pusty stos S_1 oznacza, że nie ma takiej konfiguracji, w której nie rozpatrywano by już wszystkich alternatywnych ruchów automatu. Automat *PDAMS* nie jest w stanie podzielić wejściowego słowa na grupy symboli (tzn. dane słowo nie jest generowane przez zadaną gramatykę).

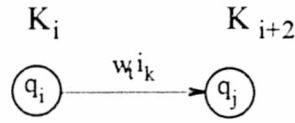
krok 6) wykonaj instrukcję i_4 i przejdź do kroku 4.

Interpretacja:

automat wraca do konfiguracji, elementy której odczytuje z pamięci stosowej.

2.3.5. Graf przejść automatu

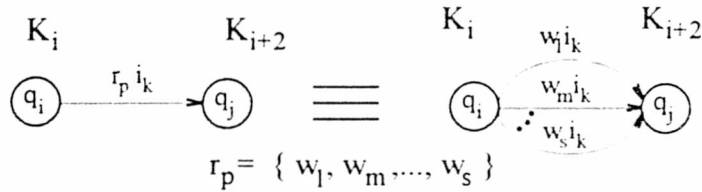
Dogodną formą reprezentacji i analizy formalnego modelu automatu *PDAMS* jest graf przejść. Graf przejść automatu jest siecią węzłów i łuków, z wyróżnionym węzłem początkowym i końcowym. Węzły odpowiadają poszczególnym stanom automatu, łuki reprezentują przejścia automatu między stanami.

Rys. 2. Łuk grafu przejść etykietowany symbolami $w_i i_k$

Przyjmujemy następujące oznaczenia łuków grafu przejść automatu *PDAMS*:

1) łuk etykietowany symbolami $w_t i_k$ ¹³ ($t = 0, 1, \dots, 8; k = 1, 2, 3, 5, 7$), gdzie w_t oznacza symbol z alfabetu wejściowego W , i_k jest symbolem instrukcji. Oznaczenie krawędzi ilustruje rys. 2. Przejście po łuku jest możliwe, gdy aktualnie czytany symbol z taśmy wejściowej jest taki sam jak symbol w_t . Przejściu towarzyszy wykonanie instrukcji i_k . Zauważmy, że tak etykietowny łuk nie reprezentuje przejścia automatu między dwoma konfiguracjami, lecz ciąg trzech konfiguracji $K_i \Rightarrow K_{i+1} \Rightarrow K_{i+2}$, przy czym, zgodnie z rys. 2, konfiguracja K_i oznacza konfigurację automatu w stanie q_i , konfiguracja K_{i+1} osiągana jest przez automat poprzez wykonanie instrukcji i_k , z konfiguracji K_{i+1} przechodzi automat do konfiguracji K_{i+2} w wyniku zmiany stanu ze stanu q_i na stan q_j .

Uwaga: w przypadku instrukcji i_5 , która jest instrukcją z parametrami, etykieta tego typu łuku ma postać $w_t i_5 q_k l_p$, gdzie q_k i l_p są parametrami instrukcji i_5 .

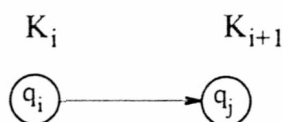
Rys. 3. Łuk grafu przejść etykietowany symbolami $r_p i_k$

2) łuk etykietowany symbolami $r_p i_k$, ($p = 1, \dots, 8; k = 1, 2, 3, 5, 7$), gdzie $r_p \subseteq W$ jest pewnym podzbiorem zbioru symboli alfabetu wejściowego (definicja tych zbiorów zostanie podana w dalszej części pracy), a i_k jak wyżej. Interpretacja tak etykietowanego łuku jest w zasadzie identyczna jak dla łuku etykietowanego symbolami $w_t i_k$, z tą różnicą, że przejście po łuku jest możliwe, gdy aktualnie czytany symbol z taśmy wejściowej jest taki sam jak dowolny z symboli zbioru r_p . Łuk oznaczony symbolami $r_p i_k$ należy rozpatrywać jako wiązkę łuków etykietowanych symbolami $w_t i_k$ zgodnie z rys. 3.

Uwaga: w przypadku instrukcji i_5 , która jest instrukcją z parametrami, etykieta tego typu łuku ma postać $r_p i_5 q_k l_t$, gdzie q_k i l_t są parametrami instrukcji i_5 .

¹³ etykietowanie krawędzi parą symboli występuje również w grafie przejść automatu typu Mealy'ego, w którym to grafie drugi symbol reprezentuje sygnał wyjściowy automatu.

3) łuk bez etykiety, reprezentujący przejście automatu między konfiguracjami K_i i K_{i+1} różniącymi się zmianą aktualnego stanu automatu (rys. 4).

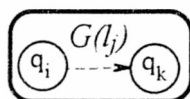


Rys. 4. Łuk grafu przejść bez etykiety

Węzłem początkowym tego typu łuku jest:

- stan $q_i \in F$, czyli stan będący stanem końcowym jednego z podautomatów: w takiej sytuacji łuk ilustruje graficznie przejście do stanu wskazywanego przez wierzchołek stosu S_4 (zob. wykonanie instrukcji i_6 w kroku 2 algorytmu działania automatu),
- stan, z którego nie ma dozwolonego ruchu dla aktualnie czytanego symbolu wejściowego: w takiej sytuacji łuk ilustruje graficznie przejście automatu do konfiguracji, której elementy odczytywane są z pamięci stosowej (zob. wykonanie instrukcji i_4 w kroku 6 algorytmu działania automatu). Dla uproszczenia grafu przejść automatu *PDAMS* zakładamy, że łuki tego typu nie będą rysowane.

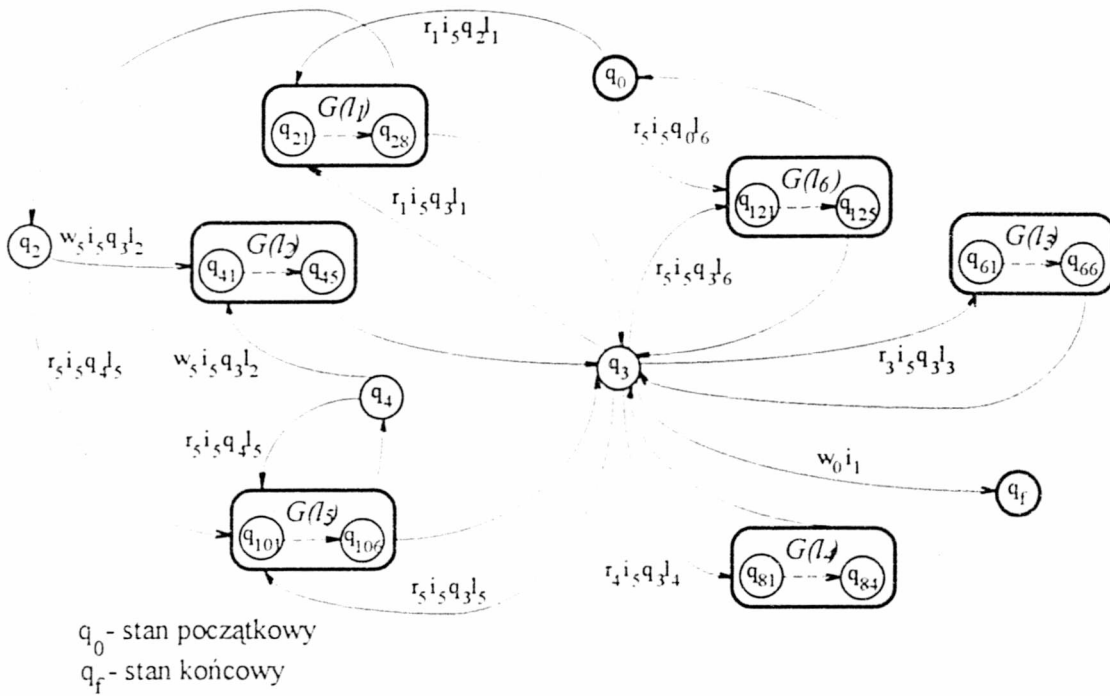
Zanim zostanie podany kompletny graf przejść automatu *PDAMS* poczynić musimy kilka uwag dotyczących samej struktury automatu. Jak wspomniano już wcześniej, z ogólnego warunku minimalnej liczby stanów w automacie wynika konieczność wyróżnienia w strukturze automatu *PDAMS* podautomatów stanowych. Dodatkowo, zbiory stanów podautomatów są rozłączne. Każdy z podautomatów stanowych akceptuje (a w konsekwencji wydziela) pewien zbiór podciągów $(w)_i$. Każdy z tych zbiorów definiuje strukturę syntaktyczną jednego z komponentów strukturalno-semantycznych na jakie dekomponujemy zdanie. Przyjmując, że w zdaniu wyróżniamy sześć typów komponentów (o strukturze syntaktycznej grupy nominalnej, werbalnej, przymiotnikowej, przysłówkowej, przyimkowej i werbalnej nieosobowej), w automacie zostaje wydzielonych sześć podautomatów stanowych, oznaczanych odpowiednio $A(l_1)$, $A(l_2)$, $A(l_3)$, $A(l_4)$, $A(l_5)$, $A(l_6)$. Stąd, automat *PDAMS* możemy traktować jako pewną sieć wyżej wymienionych podautomatów.



Rys. 5. Schematyczny rysunek podgrafu $G(l_j)$

Z powyższych uwag dotyczących struktury automatu *PDAMS* wynika, że w kompletnym grafie przejść automatu można wyróżnić sześć podgrafów¹⁴, odpowiadających poszczególnym podautomatom stanowym i oznaczanych odpowiednio $G(l_1)$, $G(l_2)$, $G(l_3)$, $G(l_4)$, $G(l_5)$, $G(l_6)$. Schematycznie podgraf $G(l_j)$ będziemy przedstawiali jak na rys. 5. Stan q_i jest stanem początkowym podautomatu $A(l_j)$, stan q_k jest jego stanem końcowym ($q_k \in F$).

Graf przejść automatu *PDAMS* z wyróżnionymi podgrafami został przedstawiony na rys. 6. Zbiory r_1 , r_3 , r_4 , r_5 są zdefiniowane następująco: $r_1 = \{w_1, w_2, w_3, w_4, w_6, w_7\}$, $r_3 = \{w_3, w_4, w_6\}$, $r_4 = \{w_3, w_6\}$, $r_5 = \{w_6, w_7\}$.



Rys. 6. Graf przejść automatu *PDAMS* z wyróżnionymi podgrafami

W komputerowej implementacji w miejsce grafu przejść stosujemy wyrażenie symboliczne [Kazimierzczak, 1978] G^+ opisujące w sposób analityczny graf:

$$G^+ = (q_0^1 (r_1 i_5 q_2 l_1 q_{21}, r_5 i_5 q_0 l_6 q_{121})^1, q_2^1 (w_5 i_5 q_3 l_2 q_{41}, r_5 i_5 q_4 l_5 q_{101})^1, q_3^1 (w_0 i_1 q_f, r_5 i_5 q_3 l_5 q_{101}, r_1 i_5 q_3 l_1 q_{21}, r_3 i_5 q_3 l_3 q_{61}, r_4 i_5 q_3 l_4 q_{81}, r_5 i_5 q_3 l_6 q_{121})^1, q_4^1 (w_5 i_5 q_3 l_2 q_{41}, r_5 i_5 q_4 l_5 q_{101})^1)^0 \quad (1)$$

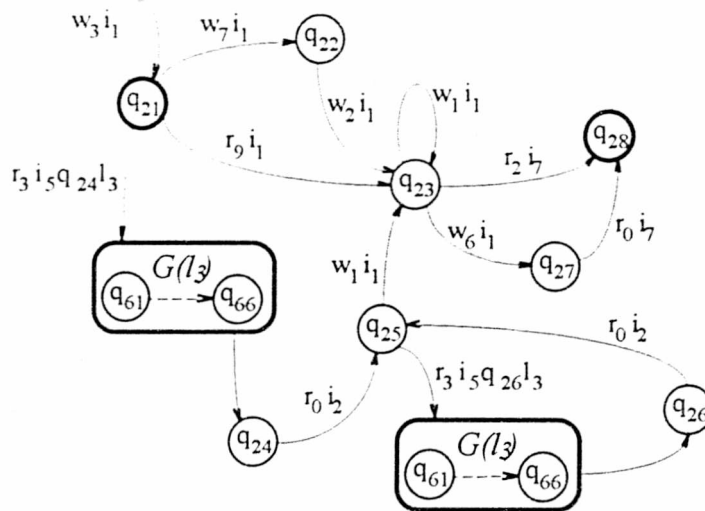
¹⁴ Podgraf jest grafem, który powstaje przez wydzielenie podzbioru węzłów i zachowanie wszystkich łuków incydentnych względem par węzłów należących do wydzielonego podzbioru [J.L.Kulikowski, 1986].

Dokładny opis algorytmu konstrukcji wyrażenia symbolicznego reprezentującego graf automatu przedstawiony został w pracach [Kazimierzczak, 1978, 1979]. W tym miejscu ograniczymy się do zinterpretowania jedynie początkowego fragmentu wyrażenia $G^+(1)^0(q_0^1(r_1i_5q_2l_1q_2l_1, \dots)$. Fragment ten oznacza, że pierwszą krawędzią wychodzącą z wierzchołka q_0 jest krawędź etykietowana symbolami $r_1i_5q_2l_1$, która to krawędź dochodzi do wierzchołka q_2l_1 . Zgodnie z uwagami poczynionymi wcześniej, przejście po tej krawędzi jest możliwe, gdy aktualnie czytany symbol z taśmy wejściowej należy do zbioru r_1 . Przejściu towarzyszy wykonanie instrukcji i_5 z dwoma parametrami: q_2 oraz l_1 .

Wyrażenie symboliczne G^+ jest fragmentem wyrażenia G^{++} (podanego w dalszej części pracy), które opisuje w sposób kompletny (tzn. definiuje również konstrukcję poszczególnych podautomatów) strukturę automatu *PDAMS*.

W pamięci komputera wyrażenie G^+ będzie pamiętane w formie rozszerzonej [Kazimierzczak, 1992] G'^+ , tzn. z rozpisaniem krawędzi typu $r_p i_k$ na reprezentowane przez nie wiązki krawędzi typu w_{ijk} :

$$G'^+ = {}^0(q_0^1(w_1i_5q_2l_1q_2l_1, w_2i_5q_2l_1q_2l_1, w_3i_5q_2l_1q_2l_1, w_4i_5q_2l_1q_2l_1, w_6i_5q_2l_1q_2l_1, w_7i_5q_2l_1q_2l_1, w_6i_5q_0l_6q_1l_2l_1, w_7i_5q_0l_6q_1l_2l_1)^1, q_2^1(w_5i_5q_3l_2q_4l_1, w_6i_5q_4l_5q_1l_0l_1, w_7i_5q_4l_5q_1l_0l_1)^1, q_3^1(w_0i_1q_6, w_6i_5q_3l_5q_1l_0l_1, w_7i_5q_3l_5q_1l_0l_1, w_1i_5q_3l_1q_2l_1, w_2i_5q_3l_1q_2l_1, w_3i_5q_3l_1q_2l_1, w_4i_5q_3l_1q_2l_1, w_6i_5q_3l_1q_2l_1, w_7i_5q_3l_1q_2l_1, w_3i_5q_3l_3q_6l_1, w_4i_5q_3l_3q_6l_1, w_6i_5q_3l_3q_6l_1, w_3i_5q_3l_4q_8l_1, w_6i_5q_3l_4q_8l_1, w_6i_5q_3l_6q_1l_2l_1, w_7i_5q_3l_6q_1l_2l_1)^1, q_4^1(w_5i_5q_3l_2q_4l_1, w_6i_5q_4l_5q_1l_0l_1, w_7i_5q_4l_5q_1l_0l_1)^1)^0 \quad (2)$$



q_{21} - stan początkowy podautomatu
 q_{28} - stan końcowy podautomatu

Rys. 7. Podgraf przejść $G(l_1)$ podautomatu $A(l_1)$ wydzielającego z ciągu (w) podciąg o strukturze syntaktycznej grupy nominalnej

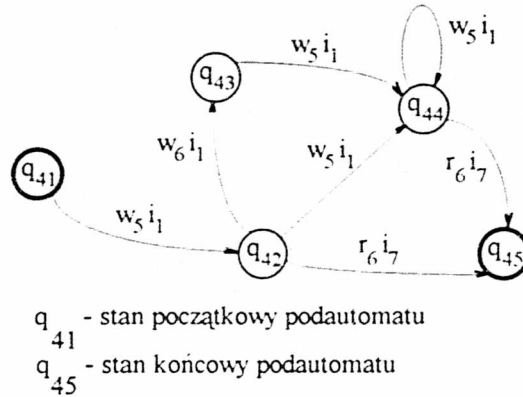
Rozważmy obecnie strukturę poszczególnych podgrafów wchodzących w skład grafu przejść automatu *PDAMS*.

Podgraf $G(l_1)$ przedstawiono na rys. 7. Zbiory r_0 , r_2 i r_9 są zdefiniowane następująco: $r_0 = W$, $r_2 = \{w_0, w_2, w_3, w_4, w_5, w_6, w_7, w_8\}$, $r_9 = \{w_1, w_2\}$. Podgraf ten określa strukturę podautomatu $A(l_1)$. Zadaniem podautomatu $A(l_1)$ jest wydzielenie z ciągu (w) tworzącego słowo wejściowe automatu podciągu definiującego komponent o strukturze syntaktycznej grupy nominalnej.

Podgraf $G(l_1)$ jest opisany analitycznie przez wyrażenie $G^+(l_1)$:

$$G^+(l_1) = {}^0(q_{21}^1(w_{31}q_{21}, w_{71}q_{22}^2(w_{21}q_{23}^3(w_{11}q_{23}, w_{61}q_{27}^4(r_{01}q_{28})^4, r_{21}q_{28})^3)^2, r_{91}q_{23}, r_{35}q_{24}l_{3q61})^1, q_{24}^1(r_{01}q_{25}^2(w_{11}q_{23}, r_{35}q_{61}l_{3q61})^2)^1, q_{26}^1(r_{01}q_{25})^1)^0 \quad (3)$$

Podautomat $A(l_2)$ wyróżnia w słowie wejściowym automatu *PDAMS* podciąg symboli $w_1w_2...$ odpowiadający komponentowi o strukturze syntaktycznej grupy werbalnej. Podgraf przejść $G(l_2)$ podautomatu $A(l_2)$ przedstawiono na rys. 8.



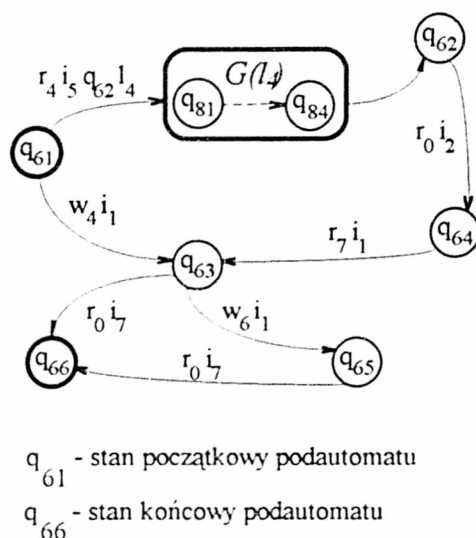
Rys. 8. Podgraf przejść $G(l_2)$ podautomatu $A(l_2)$ wydzielającego z ciągu (w) podciąg o strukturze syntaktycznej grupy werbalnej

Zbiór r_6 ma postać $r_6 = \{w_0, w_1, w_2, w_3, w_4, w_6, w_7, w_8\}$. Podgraf $G(l_2)$ jest opisany analitycznie przez wyrażenie $G^+(l_2)$:

$$G^+(l_2) = {}^0(q_{41}^1(w_{51}q_{42}^2(w_{51}q_{44}^3(w_{51}q_{44}, r_{61}q_{45})^3, w_{61}q_{43}^2(w_{51}q_{44})^2, r_{61}q_{45})^1)^0 \quad (4)$$

Podgraf $G(l_3)$ został przedstawiony na rys. 9. Zbiór r_7 ma postać $r_7 = \{w_4, w_5\}$. Podgraf ten definiuje strukturę podautomatu $A(l_3)$ odpowiedzialnego za wydzielenie ze

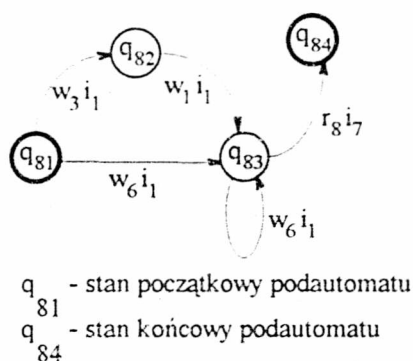
słowa wejściowego automatu *PDAMS* podciągu odpowiadającego komponentowi strukturalno-semantycznemu o strukturze syntaktycznej grupy przymiotnikowej.



Rys. 9. Podgraf przejść $G(l_3)$ podautomatu $A(l_3)$ wydzielającego z ciągu (w) podciąg o strukturze syntaktycznej grupy przymiotnikowej

Podgraf $G(l_3)$ jest opisany analitycznie przez wyrażenie $G^+(l_3)$:

$$G^+(l_3) = {}^0(q_{61}^1(r_{45}q_{62}l_4q_{81}, w_4i_1q_{63}^2(r_{07}q_{66}, w_6i_1q_{65}^3(r_{07}q_{66})^3)^2)^1, q_{62}^1(r_{02}q_{64}^2(r_{71}q_{63})^2)^1)^0 \quad (5)$$



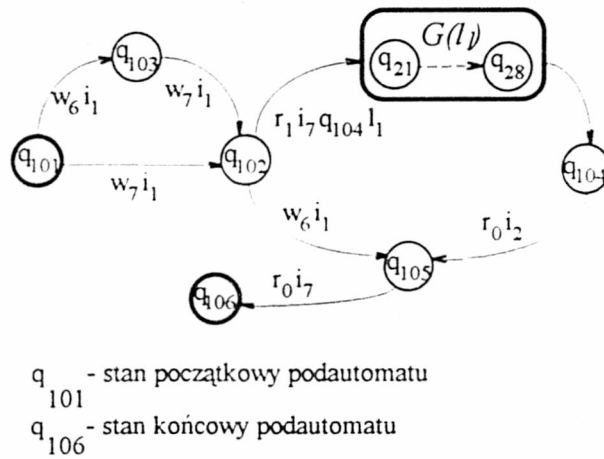
Rys. 10. Podgraf przejść $G(l_4)$ podautomatu $A(l_4)$ wydzielającego z ciągu (w) podciąg o strukturze syntaktycznej grupy przysłówkowej

Podautomat $A(l_4)$ wyróżnia w słowie wejściowym automatu *PDAMS* podciąg odpowiadający komponentowi o strukturze syntaktycznej grupy przysłówkowej. Podgraf przejść $G(l_4)$ podautomatu $A(l_4)$ przedstawiono na rys. 10. Zbiór r_8 ma postać

$r_8 = \{w_0, w_1, w_2, w_3, w_4, w_5, w_7, w_8\}$. Podgraf $G(l_4)$ jest opisany analitycznie przez wyrażenie $G^+(l_4)$:

$$G^+(l_4) = {}^0(q_{81}^1(w_{31}q_{82}^2(w_{11}q_{83}^3(w_{61}q_{83}, r_{81}q_{84})^3)^2, w_{61}q_{83})^1)^0 \quad (6)$$

Na rys. 11 przedstawiono podgraf $G(l_5)$. Podgraf ten określa strukturę podautomatu $A(l_5)$, zadaniem którego jest wydzielenie z ciągu (w) tworzącego słowo wejściowe automatu podciągu definiującego komponent o strukturze syntaktycznej grupy przyimkowej.



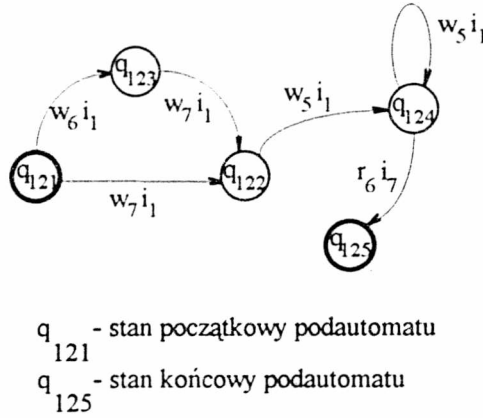
Rys. 11. Podgraf przejść $G(l_5)$ podautomatu $A(l_5)$ wydzielającego z ciągu (w) podciąg o strukturze syntaktycznej grupy przyimkowej

Podgraf $G(l_5)$ jest opisany analitycznie przez wyrażenie $G^+(l_5)$:

$$G^+(l_5) = {}^0(q_{101}^1(w_{71}q_{102}^2(w_{61}q_{105}^3(r_{01}q_{106})^3, r_{11}q_{104}q_{102})^2, w_{61}q_{103}^2(w_{71}q_{102})^2)^1, q_{104}^1(r_{02}q_{105})^1)^0 \quad (7)$$

Ostatnim podautomatem wyróżnianym w automacie *PDAMS* jest podautomat $A(l_6)$, który wydziela z ciągu (w) podciąg odpowiadający komponentowi o strukturze syntaktycznej grupy werbalnej nieosobowej. Podgraf przejść $G(l_6)$ podautomatu $A(l_6)$ przedstawiono na rys. 12. Podgraf $G(l_6)$ jest opisany analitycznie przez wyrażenie $G^+(l_6)$:

$$G^+(l_6) = {}^0(q_{121}^1(w_{71}q_{122}^2(w_{51}q_{124}^3(w_{51}q_{124}, r_{61}q_{125})^3)^2, w_{61}q_{123}^2(w_{71}q_{122})^2)^1)^0 \quad (8)$$



Rys. 12. Podgraf przejść $G(l_6)$ podautomatu $A(l_6)$ wydzielającego z ciągu (w) podciąg o strukturze syntaktycznej grupy werbaalnej nieosobowej

Wyrażenie G^{++} opisujące w sposób kompletny strukturę automatu $PDAMS$ powstaje w wyniku operacji nakładania na siebie wyrażeń G^+ (1), $G^+(l_1)$ (3), $G^+(l_2)$ (4), $G^+(l_3)$ (5), $G^+(l_4)$ (6), $G^+(l_5)$ (7), $G^+(l_6)$ (8). Operator $\oplus$ oznacza operację nakładania na siebie (pokrycia) wyrażeń symbolicznych będących jego argumentami [Kazimierzczak, 1979].

$$\begin{aligned}
 G^{++} &= G^+ \oplus G^+(l_1) \oplus G^+(l_2) \oplus G^+(l_3) \oplus G^+(l_4) \oplus G^+(l_5) \oplus G^+(l_6) = \\
 &= {}^0(q_0^1(r_{1i}q_2l_{1q21}, r_{5i}q_0l_{6q121})^1, q_2^1(w_{5i}q_3l_{2q41}, r_{5i}q_4l_{5q101})^1, q_3^1(w_{0i}q_6, \\
 &r_{5i}q_3l_{5q101}, r_{1i}q_3l_{1q21}, r_{3i}q_3l_{3q61}, r_{4i}q_3l_{4q81}, r_{5i}q_3l_{6q121})^1, q_4^1(w_{5i}q_3l_{2q41}, \\
 &r_{5i}q_4l_{5q101})^1, q_{21}^1(w_{3i}q_{21}, w_{7i}q_{22}^2(w_{2i}q_{23}^3(w_{1i}q_{23}, w_{6i}q_{27}^4(r_{0i}q_{28})^4, \\
 &r_{2i}q_{28})^3)^2, r_{9i}q_{23}, r_{3i}q_{24}l_{3q61})^1, q_{24}^1(r_{0i}q_{25}^2(w_{1i}q_{23}, r_{3i}q_{61}l_{3q61})^2)^1, \\
 &q_{26}^1(r_{0i}q_{25})^1, q_{41}^1(w_{5i}q_{42}^2(w_{5i}q_{44}^3(w_{5i}q_{44}, r_{6i}q_{45})^3, w_{6i}q_{43}^2(w_{5i}q_{44})^2, \\
 &r_{6i}q_{45})^1, q_{61}^1(r_{4i}q_{62}l_{4q81}, w_{4i}q_{63}^2(r_{0i}q_{66}, w_{6i}q_{65}^3(r_{0i}q_{66})^3)^2)^1, \\
 &q_{62}^1(r_{0i}q_{64}^2(r_{7i}q_{63})^2)^1, q_{81}^1(w_{3i}q_{82}^2(w_{1i}q_{83}^3(w_{6i}q_{83}, r_{8i}q_{84})^3)^2, w_{6i}q_{83})^1, \\
 &q_{101}^1(w_{7i}q_{102}^2(w_{6i}q_{105}^3(r_{0i}q_{106})^3, r_{1i}q_{104}l_{1q21})^2, w_{6i}q_{103}^2(w_{7i}q_{102})^2)^1, \\
 &q_{104}^1(r_{0i}q_{105})^1, q_{121}^1(w_{7i}q_{122}^2(w_{5i}q_{124}^3(w_{5i}q_{124}, r_{6i}q_{125})^3)^2, \\
 &w_{6i}q_{123}^2(w_{7i}q_{122})^2)^1)^0
 \end{aligned} \tag{9}$$

2.3.6. Dekompozycja ciągu (w) na podciągi (w)_i

Zasada działania automatu $PDAMS$ zostanie zilustrowana opowiednim przykładem. Załóżmy, że zadaniem automatu jest dekompozycja następującego ciągu symboli w_j: w₄w₁w₅w₇w₅w₀ na podciągi definiujące komponenty strukturalno-semantyczne. Ciąg ten odpowiada przykładowo zdaniu *White house is to let*. W konfiguracji początkowej automat znajduje się w stanie początkowym q₀, czyta pierwszy symbol ciągu w₄, stosy automatu są puste. Aby zwiększyć czytelność przykładu przy zapisie markerów t₁ i t₂ ze zbioru T

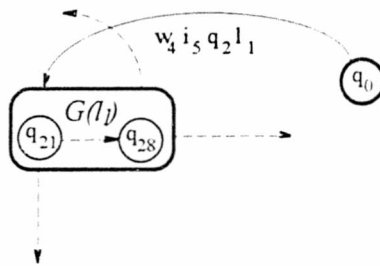
stosować będziemy odpowiednio symbol nawiasu lewego "(" oraz symbol nawiasu prawego ")".

1. Po wpływie symbolu w_4 automat przechodzi do stanu q_{21} (początkowy stan podautomatu $A(l_1)$) wykonując instrukcję i_5 z dwoma parametrami: q_2, l_1 . Instrukcja i_5 zapisuje stosy S_3 i S_4 jak niżej:

S_3 : $z_0 3 l_1 ($

S_4 : $z_0 4 q_2$

Przejście ze stanu q_0 do stanu q_{21} pod wpływem czytanej z taśmy wejściowej symbolu w_4 ($w_4 \in r_1$) ilustruje rys. 13.

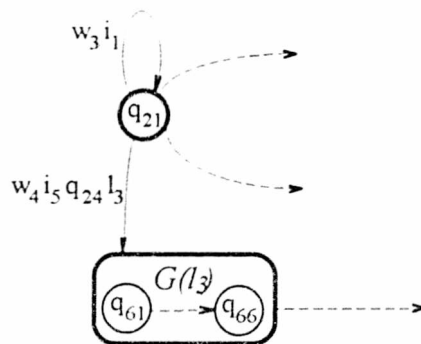


Rys. 13. Fragment grafu przejść automatu PDA_{M15}

2. Czytając symbol w_4 automat przechodzi ze stanu q_{21} do stanu q_{61} (początkowy stan podautomatu $A(l_3)$) wykonując instrukcję i_5 z dwoma parametrami: q_{24}, l_3 . Po wykonaniu instrukcji i_5 stosy S_3 i S_4 są zapisane jak niżej:

S_3 : $z_0 3 l_1 (l_3 ($

S_4 : $z_0 4 q_2 q_{24}$



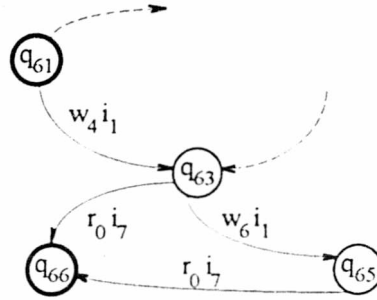
Rys. 14. Fragment podgrafu przejść $G(l_1)$

Na rys. 14 zostało przedstawione przejście automatu ze stanu q_{21} do stanu q_{61} wywołane przez symbol w_4 ($w_4 \in r_3$).

3. Po wpływie symbolu w_4 automat przechodzi do stanu q_{63} . Podczas przejścia jest wykonywana instrukcja i_1 , która na stos S_3 wstawia czytany symbol w_4 i przesuwa głowicę taśmy wejściowej pod następny symbol w_1 .

$S_3: z_031_1(l_3(w_4$

Rys. 15 ilustruje przejście automatu ze stanu q_{61} do stanu q_{63} wywołane przez symbol w_4 .



Rys. 15. Fragment podgrafu przejść $G(l_3)$

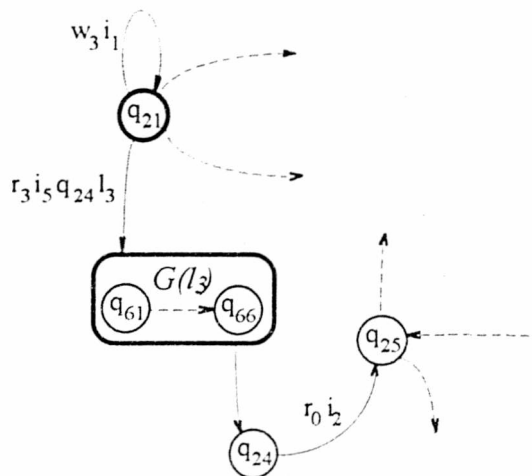
4. Pod wpływem symbolu w_1 automat przechodzi do stanu q_{66} . Przejściu towarzyszy wykonanie instrukcji i_7 . Instrukcja i_7 zapisuje na stosie S_6 zawartość stosu S_3 , a następnie marker t_2 oraz opróżnia stos S_3 .

$S_3: z_03$

$S_6: z_061_1(l_3(w_4)$

5. Stan q_{66} jest stanem końcowym podautomatu $A(l_3)$. Zgodnie z krokiem 2 algorytmu działania automatu wykonywana jest instrukcja i_6 . Instrukcja i_6 powoduje przejście automatu do stanu pobranego ze szczytu stosu S_4 . Automat przechodzi do stanu q_{24} .

$S_4: z_04q_2$



Rys. 16. Fragment podgrafu przejść $G(l_1)$

6. Ze stanu q_{24} automat przechodzi do stanu q_{25} wykonując instrukcję i_2 . Instrukcja i_2 przepisuje sekwencję symboli $l_3t_1w_4t_2$ ze szczytu stosu S_6 na stos S_3 .

$S_3: z_0l_3(w_4)$

$S_6: z_0l_1($

Na rys. 16 zostało przedstawione przejście automatu ze stanu q_{24} do stanu q_{25} .

7. Po wpływie symbolu w_1 automat przechodzi do stanu q_{23} . Podczas przejścia jest wykonywana instrukcja i_1 , która na stos S_3 wstawia czytany symbol w_1 i przesuwa głowicę taśmy wejściowej pod następny symbol w_5 .

$S_3: z_0l_3(w_4)w_1$

8. Czytając symbol w_5 automat przechodzi do stanu q_{28} wykonując instrukcję i_7 . Instrukcja i_7 zapisuje na stosie S_6 zawartość stosu S_3 , następnie marker t_2 oraz opróżnia stos S_3 .

$S_3: z_0$

$S_6: z_0l_1(l_3(w_4)w_1)$

9. Stan q_{28} jest stanem końcowym podautomatu $A(l_1)$. Zgodnie z krokiem 2 algorytmu działania automatu wykonywana jest instrukcja i_6 . Instrukcja i_6 powoduje przejście automatu do stanu pobranego ze szczytu stosu S_4 , czyli stanu q_2 .

$S_4: z_0$

10. Po wpływie symbolu w_5 automat przechodzi do stanu q_{41} (początkowy stan podautomatu $A(l_2)$) wykonując instrukcję i_5 z dwoma parametrami: q_3 , l_2 . Instrukcja i_5 zapisuje stosy S_3 i S_4 jak niżej:

$S_3: z_0l_2($

$S_4: z_0q_3$

11. Po wpływie symbolu w_5 automat przechodzi do stanu q_{42} . Podczas przejścia jest wykonywana instrukcja i_1 , która na stos S_3 wstawia czytany symbol w_5 i przesuwa głowicę taśmy wejściowej pod następny symbol w_7 .

$S_3: z_0l_2(w_5$

12. Ze stanu q_{42} automat pod wpływem symbolu w_7 przechodzi do stanu q_{45} wykonując instrukcję i_7 . Instrukcja i_7 zapisuje na stosie S_6 zawartość stosu S_3 , następnie marker t_2 oraz opróżnia stos S_3 .

$S_3: z_0$

$S_6: z_0l_1(l_3(w_4)w_1)l_2(w_5)$

13. Stan q_{45} jest stanem końcowym podautomatu $A(l_2)$. Zgodnie z krokiem 2 algorytmu działania automatu wykonywana jest instrukcja i_6 , która powoduje przejście automatu do stanu pobranego ze szczytu stosu S_4 , czyli stanu q_3 .

$S_4: z_0$

14. Czytając symbol w_7 automat ma w stanie q_3 do wyboru trzy możliwe ruchy: do stanu q_{101} , q_{121} i $q_{21}(w_7 \in r_5 \wedge w_7 \in r_1)$. Stąd, zgodnie z krokiem 4a algorytmu działania automat wybiera jeden ruch ze zbioru ruchów możliwych $K(q_a)$, założmy ruch do stanu

q_{101} , a następnie zbiór $K(q_a)$ zmniejsza o ruch wybrany. Ponieważ zbiór $K(q_a)$ nie jest zbiorem pustym, to zgodnie z krokiem 4b algorytmu działania wykonywana jest instrukcja i_3 , która powoduje zapamiętanie elementów aktualnej konfiguracji w pamięci stosowej:

$S_1: z_0l_93$

$S_2: z_02^4$

$S_5: z_05p$

$S_6: z_06l_1(l_3(w_4)w_1)l_2(w_5)p$

Instrukcja i_3 zapisuje na stosie S_1 aktualny stan automatu q_3 , na stosie S_2 liczbę przeczytanych symboli z taśmy wejściowej, na stosie S_5 p-marker i zawartość stosu S_3 (który jest pusty) oraz na stosie S_6 p-marker. Po wykonaniu instrukcji i_3 automat wykonuje wybrany ruch.

15. Ze stanu q_3 automat przechodzi do stanu q_{101} (początkowego stanu podautomatu $A(l_5)$). Przejściu towarzyszy wykonanie instrukcji i_5 z dwoma parametrami: q_3 , l_5 . Instrukcja i_5 zapisuje stosy S_3 i S_4 jak niżej:

$S_3: z_03l_5($

$S_4: z_04q_3$

16. Po wpływie symbolu w_7 automat przechodzi do stanu q_{102} . Podczas przejścia jest wykonywana instrukcja i_1 , która na stos S_3 wstawia czytany symbol w_7 i przesuwa głowicę taśmy wejściowej pod następny symbol w_5 .

$S_3: z_03l_5(w_7$

17. Automat będąc w stanie q_{102} i czytając symbol w_5 nie jest w stanie wykonać żadnego ruchu. Ponieważ stos S_1 nie jest pusty, więc zgodnie z krokiem 6 algorytmu działania automat wykonuje instrukcję i_4 , a następnie przechodzi do kroku 4 algorytmu. Instrukcja i_4 powoduje przejście automatu do konfiguracji zapamiętanej w 14-tym punkcie przykładu. Ze szczytu stosu S_1 pobierany jest symbol stanu q_3 , do którego przechodzi automat, zostaje odtworzona zawartość stosu S_3 na podstawie zawartości stosu S_5 , licząc od szczytu stosu do p-markera, głowica taśmy wejściowej przesuwa się nad czwarty symbol taśmy, gdzie numer klatki taśmy wskazywany jest przez szczyt stosu S_2 , wreszcie ze stosu S_6 usuwana jest grupa symboli licząc od szczytu stosu do p-markera (w analizowanym przykładzie po p-markerze na stosie S_6 nie ma zapisanego żadnego symbolu). Po wykonaniu instrukcji i_4 zawartość stosów S_1 , S_2 , S_3 , S_5 i S_6 jest następująca:

$S_1: z_01$

$S_2: z_02$

$S_3: z_03$

$S_5: z_05$

$S_6: z_06l_1(l_3(w_4)w_1)l_2(w_5)$

18. Zgodnie z krokiem 4a algorytmu działania automat wybiera kolejny ruch ze zbioru ruchów możliwych $K(q_a)$ ze stanu q_3 , założmy ruch do stanu q_{121} , a następnie zbiór $K(q_a)$

zmniejsza o ruch wybrany. Ponieważ zbiór $K(q_a)$ nie jest zbiorem pustym, to zgodnie z krokiem 4b algorytmu działania wykonywana jest instrukcja i_3 , która powoduje zapamiętanie elementów aktualnej konfiguracji w pamięci stosowej:

$S_1: z_0l_3$

$S_2: z_0z_4^4$

$S_5: z_0sp$

$S_6: z_0l_1(l_3(w_4)w_1)l_2(w_5)p$

Po wykonaniu instrukcji i_3 automat wykonuje wybrany ruch do stanu q_{121} .

19. Ze stanu q_3 automat przechodzi do stanu q_{121} (początkowego stanu podautomatu $A(l_6)$). Przejściu towarzyszy wykonanie instrukcji i_5 z dwoma parametrami: q_3 , l_6 . Instrukcja i_5 zapisuje stosy S_3 i S_4 jak niżej:

$S_3: z_0l_6($

$S_4: z_0q_3$

20. Po wpływie symbolu w_7 automat przechodzi do stanu q_{122} . Podczas przejścia jest wykonywana instrukcja i_1 , która na stos S_3 wstawia czytany symbol w_7 i przesuwą głowicę taśmy wejściowej pod następny symbol w_5 .

$S_3: z_0l_6(w_7$

21. Czytając symbol w_5 automat przechodzi do stanu q_{124} . Przejściu towarzyszy wykonanie instrukcji i_1 , która na stos S_3 wstawia czytany symbol w_5 i przesuwą głowicę taśmy wejściowej pod następny symbol w_0 .

$S_3: z_0l_5(w_7w_5$

22. Pod wpływem symbolu w_0 automat przechodzi do stanu q_{125} . Przejściu towarzyszy wykonanie instrukcji i_7 . Instrukcja i_7 zapisuje na stosie S_6 zawartość stosu S_3 , a następnie marker t_2 oraz opróżnia stos S_3 .

$S_3: z_0$

$S_6: z_0l_1(l_3(w_4)w_1)l_2(w_5)pl_5(w_7w_5)$

23. Stan q_{125} jest stanem końcowym podautomatu $A(l_6)$. Zgodnie z krokiem 2 algorytmu działania automatu wykonywana jest instrukcja i_6 , która powoduje przejście automatu do stanu pobranego ze szczytu stosu S_4 , czyli stanu q_3 .

$S_4: z_04$

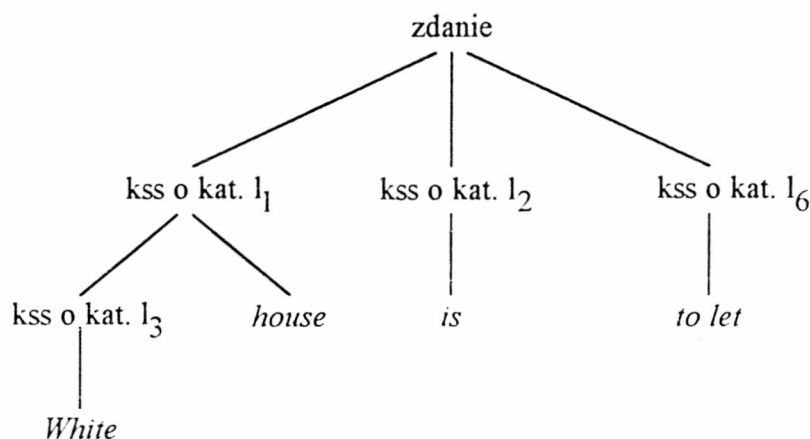
24. Pod wpływem symbolu w_0 automat przechodzi do stanu końcowego q_f . Podczas przejścia jest wykonywana instrukcja i_1 , która na stos S_3 wstawia czytany symbol w_0 i przesuwą głowicę taśmy wejściowej poza symbol w_0 .

$S_3: z_03w_0$

Zgodnie z krokiem 1 algorytmu działania automat po osiągnięciu stanu końcowego q_f kończy działanie. Na stosie S_6 zapisany jest ciąg (w) zdekomponowany na podciągi.

$S_6: z_0l_1(l_3(w_4)w_1)l_2(w_5)pl_5(w_7w_5)$

Zauważmy, że treść stosu S_6 , pomijając symbole z_{06} i p-marker, może być interpretowana jako liniowy zapis drzewa wyvodu zdania *White house is to let*. Wystarczy jedynie w miejsce symboli w_j podstawić odpowiadające im słowa zdania, natomiast zawartość odpowiadających sobie nawiasów () traktować jako definicję komponentu strukturalno-semantycznego o kategorii symbolicznie oznaczanej przez poprzedzający parę nawiasów symbol l_i (l_1 -grupa nominalna, l_2 -grupa werbalna, l_3 -grupa przymiotnikowa, l_4 -grupa przysłówkowa, l_5 -grupa przyimkowa, l_6 -grupa werbalna nieosobowa).



Rys. 17. Drzewo wyvodu zdania *White house is to let*

Na rys. 17 przedstawiono drzewo wyvodu zdania *White house is to let* otrzymanego na podstawie analizy treści stosu S_6 . Skrót kss oznacza komponent strukturalno-semantyczny.

2.4. Realizacja programowa automatu dekomponującego ciąg (w) na podciągi (w);

Automat *PDAMS* może być zrealizowany w formie sprzętowej lub programowej. W niniejszej pracy przedstawiono realizację programową automatu. Realizacja programowa sprowadza się do zapamiętania modelu formalnego automatu *PDAMS* w pamięci komputera i opracowaniu programu działającego na tym modelu. Program taki napisano w języku programowania C.

Graf przejść automatu *PDAMS* jest reprezentowany i pamiętany w postaci wyrażenia symbolicznego $G^{++}(9)$. W pracach J. Kazimierczka [1985, 1987, 1989, 1990, 1991, 1993] każdemu wyrażeniu symbolicznemu towarzyszy odpowiednia tabela, która zawiera asocjacje pomiędzy symbolami reprezentującymi wierzchołki opisywanego przez wyrażenie grafu a ich lokalizacjami w wyrażeniu. Zastosowanie tego typu tabel skraca średni czas wyszukiwania wierzchołka w wyrażeniu symbolicznym n -razy, gdzie n jest liczbą wierzchołków opisywanego grafu [Foster, 1976]. W realizacji programowej automatu

PDAMS zrezygnowano z tabeli wiążącej symbole wierzchołków grafu przejść automatu z ich adresami w wyrażeniu symbolicznym. Miejsce położenia symbolu reprezentującego poszukiwany wierzchołek w grafie przejść zwraca funkcja *szukaj*.

W realizacji komputerowej automatu wyrażenie symboliczne G^{++} jest transformowane do formy rozszerzonej G'^{++} , w której symbol r_p pojawiający się w wyrażeniu G^{++} i reprezentujący zbiór symboli w_j jest zastępowany poprzez symbole w_j należące do tego zbioru. Dodatkowo, w zapisie krawędzi postaci $w_i5q_jl_k$ w wyrażeniu G'^{++} pomija się drugi parametr l_t instrukcji i_5 . Parametr l_t jest wypracowywany przez funkcję *dajL*, która wykorzystuje jednoznaczne przyprządkowanie parametru l_t do stanu q_k ($q_{21} \rightarrow l_1$, $q_{41} \rightarrow l_2$, $q_{61} \rightarrow l_3$, $q_{81} \rightarrow l_4$, $q_{101} \rightarrow l_5$, $q_{121} \rightarrow l_6$). Stan końcowy automatu oznaczony jest symbolem q_{10} . Fragment wyrażenia G'^{++} ma postać jak niżej:

$$G'^{++} = {}^0(q_0^1(w_{15}q_{221}, w_{25}q_{221}, w_{35}q_{221}, w_{45}q_{221}, w_{65}q_{221}, w_{75}q_{221}, w_{65}q_{09121}, w_{75}q_{09121})^1, q_2^1(w_{55}q_{3441}, w_{65}q_{44101}, w_{75}q_{44101})^1, q_3^1(w_{01}q_{10}, w_{65}q_{315}q_{101}, \dots, q_{121}^1(w_{71}q_{122}^2(w_{51}q_{124}^3(w_{51}q_{124}, \dots$$
 (10)

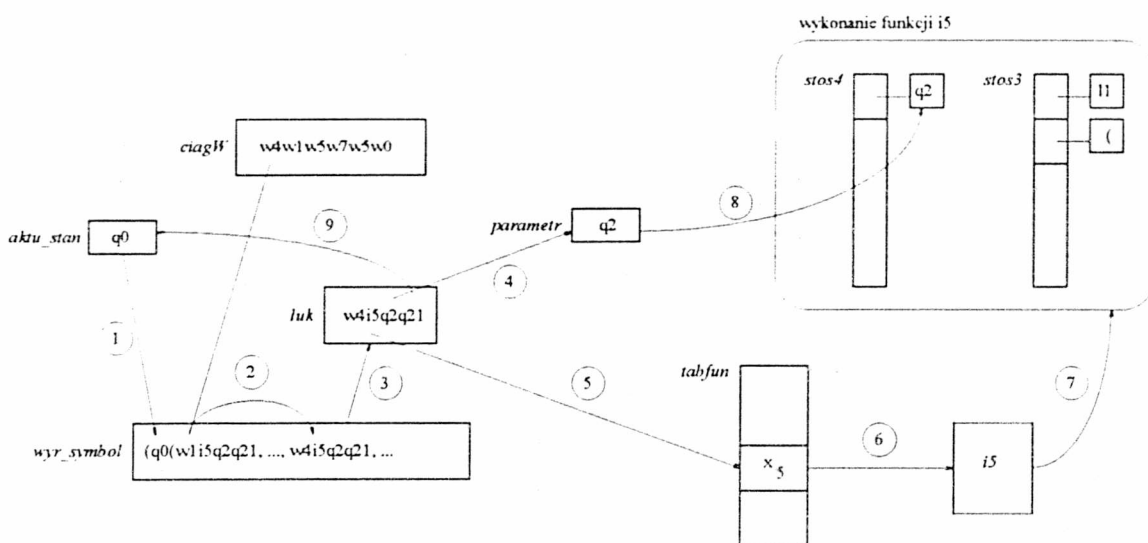
Wyrażenie symboliczne G'^{++} (10) pamiętane jest w postaci znakowej, rezydującej w spójnym obszarze pamięci wskazywanym przez wskaźnik znakowy *wyr_symbol*.

Wyrażenie symboliczne G'^{++} jednoznacznie reprezentuje działanie automatu *PDAMS*. Należy jednak zwrócić uwagę, że wyrażenie to jest jedynie elementem statycznym w pamięci komputera i nie realizuje działania automatu. Aby komputer mógł swoim działaniem reprezentować działanie automatu *PDAMS*, w pamięci komputera musi znajdować się odpowiedni program działający na wyrażeniu. Dodatkowo, konieczna jest w pamięci komputera obecność pewnych zarezerwowanych obszarów dla reprezentacji pozostałych elementów formalnego modelu automatu. Obecnie nastąpi omówienie metody ich komputerowej reprezentacji.

Każdy stos automatu reprezentowany jest przez tablicę wskaźników *stosi* ($i = 1, \dots, 6$). Szczyt stosu wskazywany jest przez indeks *wierzchSi* ($i = 1, \dots, 6$). Słowo wejściowe automatu zapisane na taśmie wejściowej reprezentowane jest przez tablicę znakową *ciagW*. Tablica znakowa *ciagW* powstaje jako rezultat odpowiedzi udzielanych przez użytkownika na pytania stawiane przez program, a dotyczące kategorii leksykalnych poszczególnych słów tworzących dekomponowane zdanie wejściowe. Aktualne położenie głowicy taśmy wejściowej określa indeks *glowicaTW*. Każdemu przejściu automatu między stanami towarzyszy wykonanie jakiejś instrukcji i_j . Każda instrukcja automatu realizowana jest jako odpowiednia funkcja o nazwie ij ($j = 1, \dots, 7$), która operuje na taśmie wejściowej i stosach automatu zgodnie ze swoją definicją podaną w rozdziale 2.3.3. W tablicy wskaźników do funkcji *tabfun* j -ty jej element zawiera adres funkcji ij . Jeżeli automat jest w konfiguracji, z której ma więcej niż jeden możliwy ruch, to ze zbioru ruchów

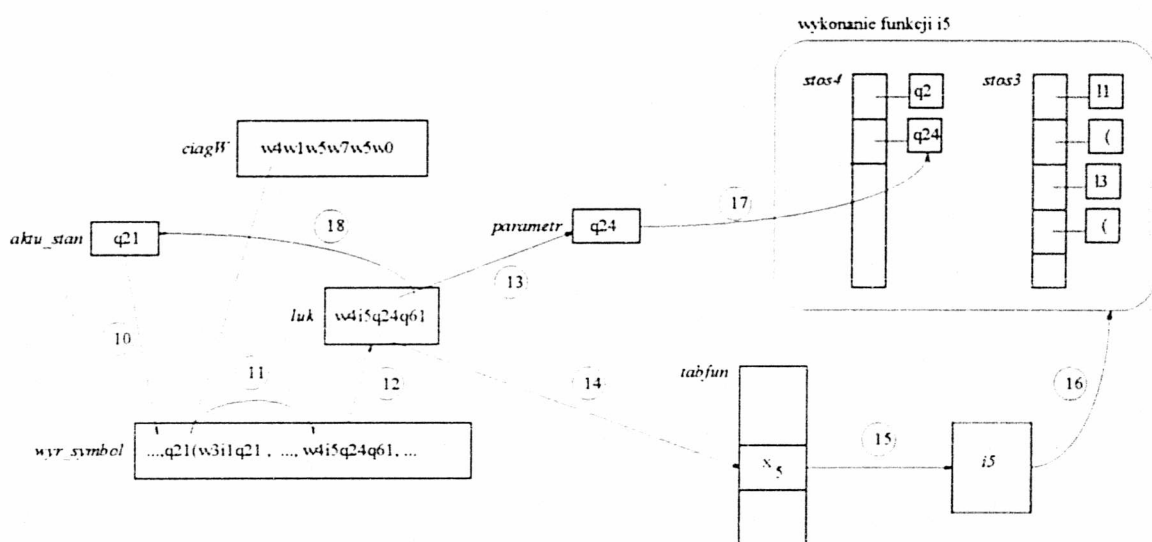
alternatywnych wybiera jeden (pierwszy, zgodnie z kolejnością zapisu alternatywnych krawędzi w wyrażeniu symbolicznym G'^{++}). Pozostałe alternatywne ruchy zapamiętuje w formie ciągu łuków na szczycie dodatkowego stosu *alterlisty*. Wierzchołek stosu wskazywany jest przez indeks *wierzch_al*. Jeżeli wybrany ruch prowadzi automat do konfiguracji, w której nie ma możliwości wykonania żadnego ruchu dla aktualnie czytanego symbolu wejściowego w_j , to automat, po wykonaniu funkcji *i4* sprowadzającej automat do ostatniej konfiguracji alternatywnych ruchów, pobiera ze szczytu stosu *alterlisty* kolejną krawędź. Wybrana krawędź przeprowadza automat do następnego stanu (krok 6, a następnie krok 4a algorytmu działania automatu). Symbol aktualnego stanu automatu pamiętany jest w obszarze pamięci wskazywanym przez wskaźnik znakowy *aktu_stan*. Aktualny parametr q_j instrukcji *i5* pamiętany jest w tablicy znakowej *parametr*, natomiast krawędź, po której realizowane jest przejście w tablicy znakowej *luk*.

Sposób reprezentacji automatu *PDAMS* w pamięci komputera zostanie wyjaśniony poprzez podział przykładowego słowa wejściowego $w_4w_1w_5w_7w_5w_0$, który to podział analizowany był już w rozdziale 2.3.6 pod kątem funkcjonowania formalnego modelu automatu. Dekompozycja słowa zilustrowana jest przez rysunki 18, 19, 20, 21, 22, które przedstawiają schematycznie organizację pamięci komputera w czasie działania programowej realizacji automatu. Dla uproszczenia przykładu zostanie przeanalizowany jedynie proces wyodrębnienia ze słowa wejściowego - ciągu (w) pierwszego pod słowa - pociągu (w)₁ odpowiadającego komponentowi o strukturze grupy przymiotnikowej. Poszczególne działania wykonywane przez komputer zostały oznaczone przez kolejne liczby otoczone okręgami.



Rys. 18. Komputerowa realizacja przejścia programowego wariantu automatu *PDAMS* ze stanu początkowego q_0 do stanu q_{21}

Na rys. 18 przedstawione zostało przejście automatu realizowanego programowo ze stanu początkowego q_0 do stanu q_{21} , który jest początkowym stanem podautomatu $A(1)$. W kroku 1 poszukiwane jest w wyrażeniu G^{+++} (10) wystąpienie ciągu znaków "q0(". W następnej kolejności poszukiwane jest w wyrażeniu G^{+++} po nawiasie otwierającym "(" wystąpienie takich termów $w_{ji}p_{qk}$ (a w przypadku instrukcji i_5 termów postaci $w_{ji}q_{tqk}$), w których występuje symbol wskazywany przez początek obszaru pamięci o nazwie *ciagW*, czyli symbol w_4 (krok 2). Znaleziony term $w_4i_5q_2q_{21}$ zapisywany jest w tablicy *luk* (krok 3). W kroku 4 do tablicy o nazwie *parametr* zapisywany jest ciąg znaków "q2", który określa pierwszy parametr instrukcji i_5 w termie $w_4i_5q_2q_{21}$. W kolejnym kroku pobierany jest indeks instrukcji z tablicy *luk* (indeksem tym jest liczba 5). Indeks wskazuje na wiersz tablicy *tabfun*, w którym znajduje się adres funkcji realizującej instrukcję i_5 (krok 5 i 6). Funkcja i_5 rezerwuje pamięć dla ciągu znaków znajdujących się w tablicy *parametr* ("q2"), kopiuje do niego zawartość tablicy *parametr*, a następnie w wierszu tablicy wskaźników *stos4* wskazywanym przez indeks *wierzchS4* umieszcza adres zarezerwowanej pamięci (krok 8). Indeks *wierzchS4* jest zwiększany o jeden.



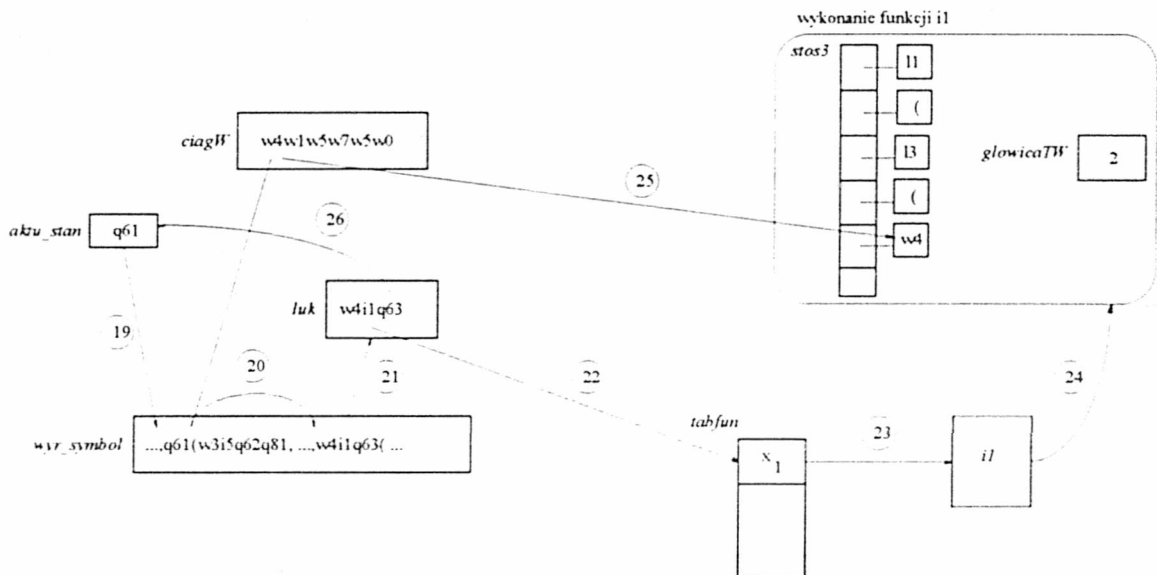
Rys. 19. Komputerowa realizacja przejścia programowego wariantu automatu *PDAMS* ze stanu q_{21} do stanu q_{61}

W następnej kolejności funkcja i_5 rezerwuje pamięć i kopiuje do niej ciąg znaków "11", który to ciąg jest zwracany przez funkcję *dajL* dla argumentu "q21". Adres zarezerwowanej pamięci jest umieszczony w wierszu tablicy wskaźników *stos3* wskazywanym przez indeks *wierzchS3*. Indeks *wierzchS3* jest zwiększany o jeden. W kolejnym wierszu tablicy *stos3*, wskazywanym przez indeks *wierzchS3*, zapisywany jest adres obszaru pamięci zawierającego znak "("). Znak ten reprezentuje marker t_1 ze zbioru T . Indeks *wierzchS3* jest zwiększany o jeden. W kroku 9 z tablicy *luk* pobierany jest ciąg

znaków "q21", który zostaje przepisany do tablicy *aktu_stan*. Krok 9 reprezentuje zmianę stanu automatu ze stanu q_0 do stanu q_{21} .

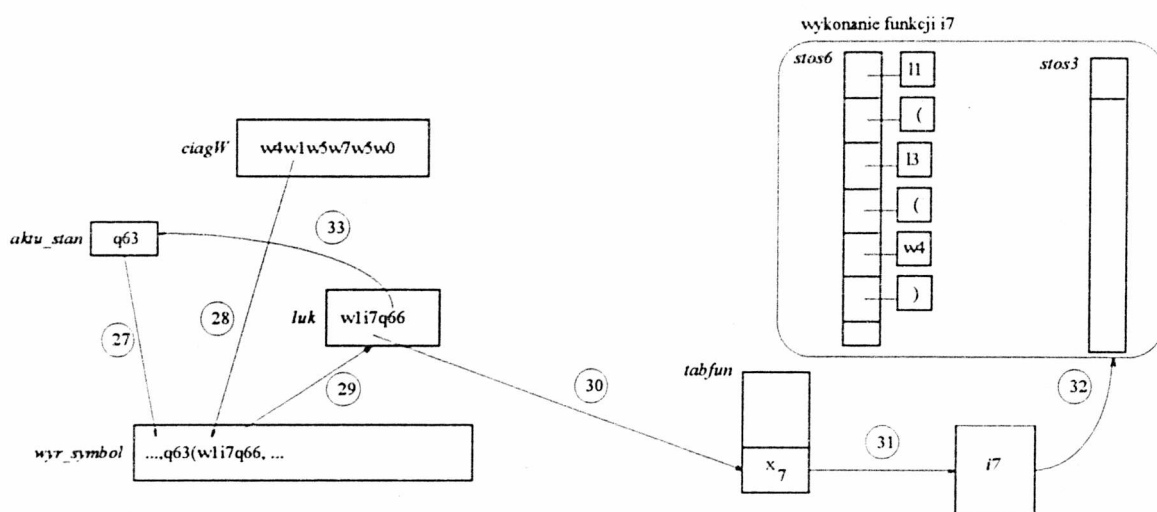
Ze stanu q_{21} automat przechodzi do stanu q_{61} , który jest początkowym stanem podautomatu $A(l_3)$. Przejściu towarzyszy wykonanie instrukcji i_5 z dwoma parametrami: symbolem stanu q_{24} oraz markerem l_3 . Kolejne kroki 10 - 18 wykonywane przez programową realizację automatu przedstawiono na rys. 19.

Na rys. 20 zilustrowano przejście automatu z aktualnego stanu q_{61} (reprezentowanego przez ciąg znaków "q61" w tablicy *aktu_stan*) do stanu q_{63} . Podczas przejścia wykonywana jest funkcja $i1$, która zapisuje w kolejnym wierszu tablicy *stos3* adres obszaru pamięci zawierającego ciąg znaków "w4" reprezentujący pierwszy symbol taśmy wejściowej (krok 25). Następnie funkcja $i1$ nadaje indeksowi *glowicaTW* wartość 2. Indeks *glowicaTW* wskazuje pierwszy znak aktualnie czytanego ciągu znaków "w_" w tablicy *ciagW*.



Rys. 20. Komputerowa realizacja przejścia programowego wariantu automatu *PDAMS* ze stanu q_{61} do stanu q_{63}

Pod wpływem symbolu w_1 automat przechodzi ze stanu q_{63} do stanu q_{66} . Przejściu towarzyszy wykonanie instrukcji i_7 . W komputerowej realizacji automatu podczas zmiany zawartości tablicy *aktu_stan* z ciągu znaków "q63" na ciąg "q66" wykonywana jest funkcja i_7 . Proces ten ilustruje rys. 21.

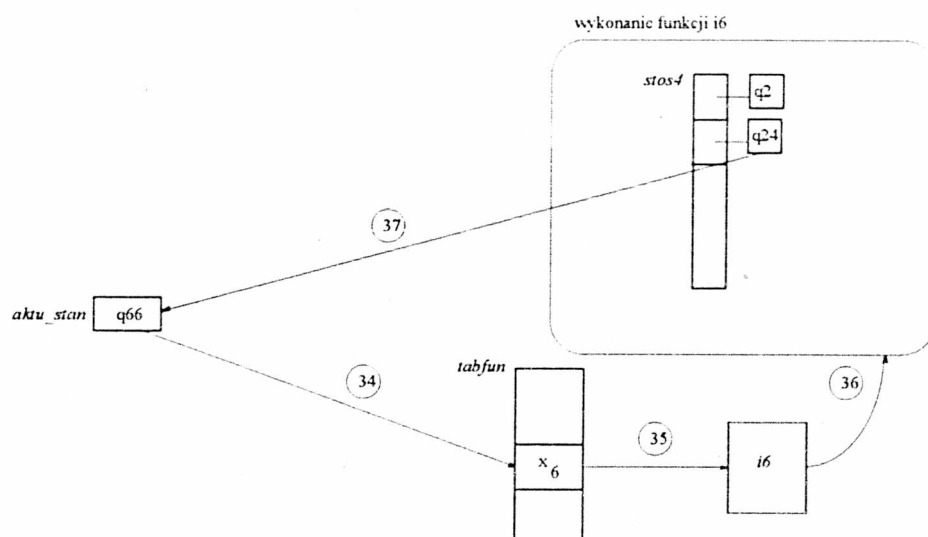


Rys. 21. Komputerowa realizacja przejścia programowego wariantu automatu *PDAMS* ze stanu q_{63} do stanu q_{66}

Funkcja i_7 przepisuje zawartość tablicy $stos3$ w tablicę $stos6$ dopisując w pierwszym wolnym wierszu tablicy $stos6$ adres obszaru pamięci zapisanego przez znak $)$). Znak ten reprezentuje marker t_2 ze zbioru T . Obszary pamięci wskazywane przez wiersze tablicy $stos3$ są zwalniane, a indeks $wierzchS3$ ustawiany jest na pierwszy element tablicy.

Aktualnym stanem automatu jest stan q_{66} . Stan ten jest stanem końcowym podautomatu $A(l_3)$ ($q_{66} \in F$).

Zgodnie z krokiem 2 algorytmu działania automatu w chwili osiągnięcia stanu końcowego podautomatu wykonywana jest instrukcja i_6 .



Rys. 22. Komputerowa realizacja przejścia programowego wariantu automatu *PDAMS* ze stanu q_{66} do stanu q_{24}

W komputerowej realizacji automatu po zapisie tablicy *aktu_stan* nowym ciągiem znaków wywoływana jest funkcja *czyf*, która sprawdza, czy ciąg znaków reprezentujący aktualny stan automatu jest tożsamy z ciągiem odpowiadającym końcowemu stanowi jednego z podautomatów. Przy odpowiedzi twierdzącej wykonywana jest funkcja *i6*. Sytuację tą ilustruje rys. 22. Funkcja *i6* przepisuje zawartość pamięci adresowanej wierszem tablicy *stos4* i wskazywanym przez indeks *wierzchS4* do tablicy *aktu_stan*. Obszar pamięci wskazywany przez ten wiersz jest zwalniany, a indeks *wierzchS4* jest zmniejszany o jeden. Przepisany do tablicy *aktu_stan* ciąg znaków "q24" reprezentuje aktualny stan automatu.

W chwili osiągnięcia stanu q_{24} na stosie S_6 reprezentowanym w pamięci komputera przez tablicę wskaźników *stos6* zapisany jest następujący ciąg znaków "l1(l3(w4)". Przyjmując, że nawiasy okrągłe wydzielają komponent strukturalno-semantyczny, a poprzedzający lewy nawias marker l_1 określa kategorię syntaktyczną komponentu, w ciągu "l1(l3(w4)" można wyróżnić komponent o strukturze grupy przymiotnikowej (symbol l3), który jest tworzony przez wyraz o kategorii leksykalnej w4 (przymiotnik). Co więcej, komponent ten wchodzi w skład innego komponentu o kategorii grupy nominalnej (symbol l1). Zadanie wyodrębnienia poszczególnych komponentów strukturalno-semantycznych na podstawie zawartości tablicy *stos6* realizuje funkcja *tworz_kss*.

Tabulogram programu AUTOMAT, stanowiącego jeden z elementów realizacji programowej automatu *PDAMS*, zamieszczono w załączniku A niniejszej pracy. Tekst programu rozdzielony jest na trzy pliki źródłowe: *automat.h*, *automat.cpp* i *menu.cpp*.

2.5. Opis funkcjonalny programu AUTOMAT

Program AUTOMAT został napisany przy użyciu pakietu programowego BORLAND C ++ 3.0 i biblioteki Turbo Vision.

Ekran programu jest podzielony na trzy części: wiersz menu (pierwszy wiersz), wiersz statusu (wiersz ostatni) oraz część roboczą ekranu (centrum ekranu). Wiersz menu umożliwia poruszanie się po dwupoziomowym menu, wiersz statusu zawiera informację o podstawowych klawiszach szybkiego reagowania, w części roboczej wyświetlane są okna tekstowe i dialogowe. Okno tekstowe w programie AUTOMAT składa się z ramki zawierającej tekst w nagłówku, piktogramów zamykania i powiększania rozmiarów, poprzeczek przesuwu zawartości okna oraz tekstu wyświetlanego wewnątrz okna. W oknach dialogowych znajdują się przyciski - ramki z poleceniami, których wybranie powoduje wykonanie jakiejś operacji oraz albo wiersz wprowadzania danych, albo podokienka z opcjami.

W programie wykorzystuje się następujące klawisze:

- <↵ Enter> - wybór podświetlonej opcji menu lub opcji podokienka w oknie dialogowym; efekt wyboru opcji menu uzyskuje się również poprzez wciśnięcie wyróżnionego klawisza w opisie opcji lub naciśnięcie lewego przycisku myszki,
- <Tab> - poruszanie się wewnątrz okienka dialogowego,
- <Esc> - wyjście do menu poziomu wyższego lub zamknięcie okna dialogowego,
- <←>, <→> - poruszanie się po opcjach menu w układzie poziomym,
- <↑>, <↓> - poruszanie się po opcjach menu w układzie pionowym,
- <Home> - przejście do pierwszej opcji menu w układzie pionowym,
- <End> - przejście do ostatniej opcji menu w układzie pionowym.

W linii statusowej programu znajduje się opis następujących klawiszy szybkiego reagowania:

- <F1> - wyświetlenie okna tekstowego z tekstem informacji pomocniczej o programie,
- <F10> - przejście do menu głównego,
- <Alt-W> - wyjście z programu.

Funkcję klawisza szybkiego reagowania w wierszu statusowym można również wywołać poprzez wybranie myszką tekstu opisującego klawisz.

Menu główne ma układ poziomy i składa się z następujących opcji:

- ≡ - krótka informacja o programie, zmiana liczby wierszy ekranu, wyjście do DOS'a,
- Zdanie - wprowadzanie nowego zdania i kategorii leksykalnych słów tworzących zdanie,
- Podział - uruchomienie automatu z możliwością pracy krokowej, przegląd uzyskanych wyników,
- Okno - otwieranie, zamykanie, powiększanie poszczególnych okienek tekstowych w części roboczej ekranu,
- Wyjście - wyjście z programu.

W kolejnych rozdziałach zostanie przedstawiony szczegółowy opis poszczególnych opcji menu głównego programu.

2.5.1. Opcja ≡

Pierwsza opcja menu głównego umożliwia:

- uzyskanie krótkiej informacji - wizytówki o programie (podopcja O Programie),
- zmianę liczby wierszy ekranu ze standardowej 25 na 43 dla karty graficznej EGA lub 50 dla karty VGA (podopcja 43/50 linii); podopcję tą można również wywołać przez kombinację klawiszy <Alt-G>.

- wyjście do poziomu systemu operacyjnego DOS z możliwością powrotu do programu (podopcja Poziom Dos'a).

2.5.2. Opcja Zdanie

Po wybraniu opcji Zdanie menu głównego w części roboczej ekranu pojawia się okno dialogowe zatytułowane "Wprowadz zdanie". W oknie tym znajduje się wiersz wprowadzania zdania oraz dwa przyciski: "Akceptacja" i "Rezygnacja". Podczas wprowadzania zdania dostępne są następujące klawisze edycyjne:

- <←>, <→> - przesuwanie kursora w obrębie tekstu,
- <Home> - przesunięcie kursora na początek tekstu,
- <End> - przesunięcie kursora na koniec tekstu,
- <Ins> - przełączanie trybu wpisywania (Insert/Overwrite),
- <Backspace> - usuwanie znaku przed kursorem,
- - usuwanie znaku na pozycji kursora,
- <↵Enter> - zakończenie edycji z akceptacją zdania,
- <Esc> - zakończenie edycji i zamknięcie okna dialogowego bez akceptacji zdania.

Akceptacja zdania powoduje zamknięcie okna dialogowego "Wprowadz zdanie" i pojawienie się na ekranie okna dialogowego zatytułowanego "Podaj kategorie syntaktyczne słowa: *", gdzie w miejscu gwiazdki pojawia się pierwsze słowo wprowadzonego zdania. W oknie dialogowym znajduje się podokienko "Kategoria syntaktyczna" z opcjami: Rzeczownik (w1), Zaimek (w2), Rodzajnik (w3), Przymiotnik (w4), Czasownik (w5), Przysłówka (w6), Przyimek (w7), Spójnik (w8) oraz dwa przyciski "Akceptacja" i "Rezygnacja". Dla podanego słowa w tytule okna określa się kategorię leksykalną (symbol w_j) albo poprzez naciśnięcie odpowiedniego wyróżnionego klawisza albo poprzez wybranie odpowiedniej podopcji klawiszem <Tab>. Potwierdzenie wyboru dokonuje się klawiszem <↵Enter>. Kategorię leksykalną słowa można również podać wykorzystując myszkę. Naciśnięcie klawisza <Esc> przerywa proces określania kategorii leksykalnych słów zdania, potwierdzenie wybranej kategorii powoduje zamknięcie bieżącego okna i pojawienie się okna dialogowego dla kolejnego słowa wprowadzonego zdania. Proces podawania kategorii leksykalnych dla słów wchodzących w skład zdania kończy się z chwilą określenia kategorii dla ostatniego słowa w zdaniu.

2.5.3. Opcja Podzial

Wybranie opcji Podzial menu głównego powinno być poprzedzone wprowadzeniem zdania i określeniem kategorii leksykalnych słów tworzących zdanie. Na opcję tą składają się trzy podopcje: Dziel, Wyniki, Opcje...

Podopcja Opcje... umożliwia zmianę trybu pracy działania automatu. Po jej wybraniu na ekranie pojawia się okno dialogowe "Wybierz tryb pracy". W oknie dialogowym znajduje się podokienko "Tryb pracy" z opcjami: Krokowa, Ciągła oraz dwa przyciski "Akceptacja" i "Rezygnacja". Zasada wyboru opcji w oknie dialogowym jest identyczna jak w przypadku określania kategorii leksykalnej słowa. Krokowy tryb pracy umożliwia śledzenie przechodzenia automatu *PDAMS* między stanami, zmianę zawartości stosów automatu, wykonywane instrukcje automatu oraz ruch głowicy taśmy wejściowej.

Podopcja Dziel powoduje uruchomienie automatu. W części roboczej ekranu pojawia się pięć okien tekstowych zatytułowanych kolejno "Tasma Wejsciowa", "Zawartosc Stosow", "Przejscia ze stanu do stanu", "Instrukcja" i "Alternatywne przejścia". W oknie "Tasma Wejsciowa" wypisany jest ciąg (w) skojarzony z wprowadzonym zdaniem. Pod ciągiem znajduje się znak "▲", który symbolizuje głowicę czytającą taśmy wejściowej. Aktualna pozycja znaku odpowiada położeniu głowicy czytającej. Okno "Zawartosc Stosow" demonstruje zawartość stosów $S_1, \dots, S_6$ w bieżącej konfiguracji automatu. W oknie "Instrukcja" wyświetlana jest aktualnie wykonywana przez automat instrukcja, w przypadku instrukcji i_5 dodatkowo jej pierwszy parametr q_t . Jeżeli w danej konfiguracji automat ma więcej niż jedno możliwe przejście, to w oknie "Alternatywne przejścia" wypisane są alternatywne przejścia automatu w formie termów $w_{jip}q_k$ lub w przypadku instrukcji i_5 w postaci $w_{ji5q_t}q_k$. Okno tekstowe "Przejscia ze stanu do stanu" zawiera informację stanowiącą swego rodzaju historię działania automatu. W oknie tym w kolejnych wierszach zapisywany jest numer aktualnego kroku automatu, stan, w którym automat się znajduje, krawędź, po której następuje przejście (w formie termu $w_{jip}q_k$ lub $w_{ji5q_t}q_k$) i stan, do którego automat przechodzi. Z chwilą osiągnięcia przez automat stanu końcowego q_{10} , w ostatnim wierszu wypisywany jest tekst "-> Automat osiagnal stan koncowy q_{10} <-". W przypadku pracy krokowej po każdej zmianie stanu przez automat następuje zatrzymanie jego działania do momentu naciśnięcia dowolnego klawisza.

Podopcja Wyniki uaktywnia okno tekstowe zawierające wyniki działania automatu. Okno podaje informację o wprowadzonym zdaniu, skojarzonym ze zdaniem ciągiem (w), podziale ciągu (w) na komponenty, definicje poszczególnych komponentów strukturalno-semantycznych. Zawartość okna dla zdania *Chocolate is poured into moulds at a carefully regulated temperature* jest następująca:

ZDANIE WEJSCIOWE (s):

Chocolate is poured into moulds at a carefully regulated temperature.

SKOJARZONY ZE ZDANIEM (s) CIĄG SYMBOLI (w)

w1w5w5w7w1w7w3w6w4w1w0

PODZIAŁ CIĄGU (w) NA KOMPONENTY STRUKTURALNO-SEMANTYCZNE

(w) $\leftrightarrow$ k1 k2 k3 k5

Definicja komponentów strukturalno-semantycznych

k1 (11) $\leftrightarrow$ w1 (Chocolate)

k2 (12) $\leftrightarrow$ w5 w5 (is poured)

k3 (15) $\leftrightarrow$ w7 k4 (into *)

k4 (11) $\leftrightarrow$ w1 (moulds)

k5 (15) $\leftrightarrow$ w7 k6 (at *)

k6 (11) $\leftrightarrow$ w3 k7 w1 (a * temperature)

k7 (13) $\leftrightarrow$ k8 w4 (* regulated)

k8 (14) $\leftrightarrow$ w6 (carefully)

Komponenty strukturalno-semantyczne oznaczane są symbolami k_i , w ich definicji po lewej stronie znaku " $\leftrightarrow$ " w nawiasie podany jest symbol kategorii syntaktycznej komponentu, po prawej stronie znaku " $\leftrightarrow$ " znajduje się ciąg tworzących komponent symboli w_j oraz k_t , natomiast w nawiasie podane są odpowiadające symbolom w_j słowa zdania. Gwiazdka symbolizuje wchodzący w skład komponentu inny komponent.

2.5.4. Opcja Okno

Opcja Okno menu głównego umożliwia następujące operacje na oknach tekstowych:

- otwarcie wszystkich okien (podopcja Otworz wszystkie); z podopcją tą związany jest klawisz szybkiego reagowania <F4>,
- uaktywnienie następnego okna (podopcja Następne); funkcję tą spełnia również klawisz <F6>,
- powiększenie okna (podopcja Powieksz); funkcję tą spełnia również klawisz <F5>,
- zamknięcie okna (podopcja Zamknij), funkcję tą spełnia również kombinacja klawiszy <Alt-F3>,
- zamknięcie wszystkich otwartych okien (podopcja Zamknij wszystkie).

2.5.5. Opcja Wyjście

Opcja Wyjście umożliwia wyjście z programu AUTOMAT. Wyjście z programu następuje również po naciśnięciu kombinacji klawiszy <Alt-W>.

2.6. Perspektywy rozwoju i realizacji automatu PDAMS

W niniejszej pracy przedstawiono model formalny i realizację programową automatu PDAMS, który umożliwia analizę syntaktyczną i dekompozycję na elementy bazy wiedzy zdań prostych, orzekających języka angielskiego. Przedstawiony formalizm, dzięki zredukowanemu zbiorowi dopuszczalnych instrukcji, może stanowić swego rodzaju narzędzie, pozwalające na konstruowanie parserów akceptujących bardziej złożone formy języka angielskiego, jak na przykład zdania złożone.

Zaprezentowany w pracy automat ogranicza się do znalezienia jedynie pierwszego znacznika frazowego słowa wejściowego, a w konsekwencji jednego podziału zdania na komponenty strukturalno-semantyczne.

Znalezienie właściwej dekompozycji zdania (wywodu, wyprowadzenia) decyduje o prawidłowej interpretacji semantycznej zdania. Zdanie mające więcej niż jedno wyprowadzenie jest (przy pominiętych kontekście wypowiedzi) semantycznie niejednoznaczne. Przykładem zdania niejednoznacznego semantycznie jest często cytowane w literaturze zdanie *They are flying planes*, które ma dwa wyprowadzenia. Przyjmując, że nawiasy otaczają komponenty strukturalno-semantyczne, pierwsze wyprowadzenie ma postać (*They*) (*are flying*) (*planes*), natomiast drugie (*They*) (*are*) (*flying planes*). Pierwszy sposób dekompozycji zdania powoduje, że zdanie tłumaczymy *Oni latają samolotami*, zdanie zdekomponowane w sposób drugi ma już zupełnie inną interpretację: *To są latające samoloty*.

Rozwiązaniem problemu potencjalnych niejednoznaczności dekomponowanych zdań, przy analizowaniu izolowanych wypowiedzi, może być wygenerowanie przez automat PDAMS wszystkich możliwych podziałów zdania na komponenty strukturalno-semantyczne. W tym celu konieczna się staje jedynie nieznaczna modyfikacja algorytmu działania automatu, polegająca na zmianie warunku zakończenia działania w przypadku osiągnięcia stanu końcowego (zobacz rozdział 2.3.4). Krok pierwszy przyjmie następującą postać:

krok 1) jeżeli aktualny stan $q_a = q_f$ i stos S_1 jest pusty, to zakończ działanie automatu z wynikiem pozytywnym.

Interpretacja:

pusty stos S_1 oznacza, że nie ma takiej konfiguracji, w której nie rozpatrywano by już wszystkich alternatywnych ruchów automatu (potencjalnych alternatywnych znaczników frazowych słowa wej.)

jeżeli aktualny stan $q_a = q_f$ i stos S_1 jest niepusty, to na stosie S_6 znajduje się jeden z możliwych znaczników frazowych słowa wej. Przejdź do kroku 6.

Interpretacja:

Pomimo osiągnięcia stanu końcowego automat nie rozpatrywał jeszcze wszystkich alternatywnych ruchów. W kroku 6 przechodzi do konfiguracji, której elementy zapisane są w pamięci stosowej.

Automat *PDAMS* działający według zmodyfikowanego w sposób powyższy algorytmu generuje wszystkie, syntaktycznie poprawne podziały zdania na komponenty strukturalno-semantyczne. Wybór sposobu dekompozycji zdania należy do użytkownika lub inżyniera wiedzy NLP systemu.

W pracy pominięty został również problem alternatywnych interpretacji leksykalnych słowa. Przykładowo słowo *grey* może zostać zinterpretowane jako przymiotnik (symbol w_4), rzeczownik (symbol w_1) lub czasownik (symbol w_5) [Longman..., 1987]. Narzucającym się rozwiązaniem problemu może być generowanie dla każdej możliwej interpretacji leksykalnej słowa odrębnego ciągu (w) odpowiadającego wejściowemu zdaniu, a następnie podanie kolejno każdego z wygenerowanych w ten sposób ciągów na wejście automatu *PDAMS*. Przykładowemu zdaniu *I never wear grey* odpowiadałoby sześć ciągów (w), ponieważ słowu *grey* można przypisać trzy różne symbole w_i , a słowu *wear* dwa (symbol w_1 lub symbol w_5). Przy założeniu poprawności syntaktycznej analizowanego zdania automat przynajmniej dla jednego z podanych ciągów (czyli dla jednej z możliwych interpretacji leksykalnych słowa) powinien dokonać dekompozycji zdania. W większości przypadków liczbę branych pod uwagę alternatywnych interpretacji leksykalnych słowa można by ograniczyć stosując proste reguły syntaktyczne, badające otoczenie analizowanego wyrazu (tzn. kategorie leksykalne słów sąsiednich).

W nauce o komputerach znany jest w odniesieniu do automatów skończonych fakt istnienia dualizmu pomiędzy strukturą fizyczną komputera i jego oprogramowaniem. Wybór sprzętowej lub programowej realizacji automatu skończonego zależy od funkcji, jaką ten automat ma spełniać w strukturze funkcjonalnej komputera. W niniejszej pracy opisano programową realizację automatu *PDAMS*. Automatowa reprezentacja parsera umożliwia jego sprzętową realizację w postaci układu cyfrowego o bardzo dużej skali integracji (ang. VLSI) [Chmiel, Stokłosa, 1989].

3. FORMALNY SYSTEM REPREZENTACJI WIEDZY WYRAŻONEJ W ZDANIU JĘZYKA NATURALNEGO

Podstawą formalnej reprezentacji wiedzy w KB systemu z dostępem w języku naturalnym opartej na koncepcji J. Kazimierczaka [1985, 1987, 1989, 1990, 1991, 1993] są słowniki symboli: słownik symboli reprezentujących pojedyncze słowa, słownik symboli reprezentujących komponenty strukturalno-semantyczne, słownik symboli reprezentujących zdania oraz odpowiednie wyrażenia symboliczne, które opisują relacje, jakie zachodzą pomiędzy elementami słowników. Drugim składnikiem KB, obok formalnej reprezentacji wiedzy związanej z dziedziną systemu, jest wiedza o samym języku naturalnym, w którym odbywa się komunikacja z systemem. Wiedza, którą wymaga przetwarzanie języka naturalnego, ma charakter proceduralny i jest wyrażona w formie odpowiednich reguł i procedur.

Proces transformacji wiedzy wyrażonej w zdaniu języka naturalnego (angielskiego) na postać formalną (symboliczną) poprzedza procedura gramatycznej analizy komponentu o strukturze grupy werbalnej osobowej. Procedura ta prowadzi do wyodrębnienia z komponentu rdzenia semantycznego oraz redukuje informacje o składni grupy do niezbędnego minimum, co stanowi istotną różnicę w odniesieniu do prac J. Kazimierczaka.

3.1. Formalna reprezentacja komponentu o strukturze grupy werbalnej osobowej

3.1.1 Preliminaria gramatyczne

Grupa werbalna osobowa (GWO) pełni w zdaniu języka angielskiego funkcję orzeczenia zasadniczego. Stanowi więc w swojej istocie, obok grupy nominalnej, podstawowy człon zdania [Bünting, 1985]. Dlatego też niezależnie od przyjętej teorii opisującej gramatykę języka angielskiego zawsze tworzy jedną z głównych kategorii semantycznych zdania (zwaną również rolami semantycznymi - ang. semantic roles lub przypadkami głębokimi - ang. deep cases) [Dillon, 1977].

Na podstawie reguł gramatycznych języka angielskiego [Smólska, 1976] można stwierdzić, że niezależnie od stopnia złożoności GWO, funkcję semantyczną spełnia w grupie tylko jeden czasownik (czasownik główny grupy), który występuje zawsze na ostatnim miejscu w grupie. Pozostałe formy czasownikowe (ang. auxiliary forms) pełnią jedynie funkcje gramatyczne. Głównym elementem gramatycznym w GWO jest forma osobowa czasownika, która występuje zawsze na pierwszym miejscu w grupie. Forma osobowa czasownika zależy jedynie od osoby gramatycznej i liczby podmiotu zdania. W

jednowyrazowej GWO podstawowy element gramatyczny oraz podstawowy element semantyczny zlewają się w jedno.

Na podstawie szczegółowej analizy konstrukcji GWO przyjmujemy, że każdy komponent o strukturze grupy werbalnej osobowej może być reprezentowany jako nieuporządkowany zbiór siedmiu par typu "cecha gramatyczna-wartość". Pięciu cechom: PASSIVE, PERFECT, CONTINUOUS, PAST, NEGATION przyporządkowane są wartości binarne. Jeżeli dana cecha występuje w analizowanym komponencie, to nadajemy jej wartość "1" (YES), jeżeli nie występuje to otrzymuje wartość "0" (NOT). Wartością cechy MAIN_VERB jest forma bezokolicznika czasownika głównego komponentu. Jeżeli analizowany komponent nie reprezentuje zmodalizowanej grupy to cecha MODALITY pozostaje bez przypisanej wartości, w przeciwnym wypadku wartością cechy może być jeden z następujących czasowników modalnych: *can, may, ought, will, shall, need, dare*. Warto w tym miejscu zwrócić uwagę, że jeżeli czasownikiem modalnym komponentu jest czasownik *will* lub *shall*, to możemy wnioskować, iż w zdaniu jest mowa o czasie przyszłym (ang. future). Ponieważ cechę tą można na podstawie istniejącego zbioru cech wywieść, to cecha ta nie jest wyróżniana. Przykładowemu komponentowi *will have been taken* odpowiada następujący zbiór cech gramatycznych:

MAIN_VERB:	take
PASSIVE:	YES
PERFECT:	YES
CONTINUOUS:	NOT
PAST:	NOT
NEGATION:	NOT
MODALITY:	will

3.1.2. Analiza gramatyczna komponentu

Pod pojęciem analizy gramatycznej komponentu o strukturze grupy werbalnej osobowej będziemy rozumieli odwzorowanie komponentu na zbiór siedmiu cech gramatycznych: MAIN_VERB, PASSIVE, PERFECT, CONTINUOUS, PAST, NEGATION, MODALITY. Algorytm analizy sformułowany jest w notacji pascalopodobnej w formie procedury *Analizer* i zamieszczony w załączniku B.

3.1.3. Synteza gramatyczna komponentu

Zadaniem syntezy gramatycznej komponentu o strukturze grupy werbalnej jest rekonstrukcja struktury składniowej komponentu na podstawie jego reprezentacji otrzymanej w procesie analizy.

Podczas procesu analizy tracona jest informacja o formie osobowej podstawowego gramatycznie czasownika komponentu (operatora w komponencie wielowyrazowym). Nie jest ona bowiem własnością charakterystyczną komponentu, lecz wynika z osoby gramatycznej i liczby podmiotu zdania, z którego komponent został wzięty. Stąd też formę osobową czasownika otrzymujemy w wyniku konfrontacji z podmiotem. Przedstawienie kompletnej metody ustalania formy osobowej pierwszego czasownika w rekonstruowanym komponencie wykracza poza ramy prezentowanej pracy. Tym niemniej podana zostanie przykładowa reguła ustalającą ostateczną formę osobową czasownika o formie podstawowej *be* w komponencie o cesze PAST=NOT. Zakładamy, że w istniejącym słowniku leksykalnym obok rzeczownika i zaimka występować będzie informacja dotycząca ich osoby gramatycznej i liczby. Informacja ta może być kodowana w sposób następujący: '1s', '2s', '3s' oznacza odpowiednio pierwszą, drugą i trzecią osobę liczby pojedynczej; '1p', '2p', '3p' pierwszą, drugą i trzecią osobę liczby mnogiej (jak w pracy [Allen, 1987])¹⁵.

Funkcja *Num* testuje osobę gramatyczną i liczbę podmiotu zdania, zmienna *lex* zawiera ostateczną formę czasownika:

```
if Num=1s then lex:="am"
else
if Num=3s then lex:="is"
else lex:="are";
```

Zauważmy, że algorytm syntezy może być w bardzo prosty sposób wykorzystany do zamierzonej zmiany gramatycznych cech generowanego komponentu. Przykładowo, zmiana wartości binarnej cechy PASSIVE z wartości NOT na YES spowoduje wygenerowanie komponentu już w stronie biernej.

Algorytm syntezy w formie procedury pascalopodobnej *Syntheser* jest zamieszczony w załączniku C.

3.1.4. Przesłanki reprezentacji komponentu

Jeżeli przyjmiemy reprezentację grupy zmodalizowanej za reprezentację maksymalną pod względem przechowywanej informacji, to aby nie tracić jednorodności reprezentacji można założyć, że w reprezentacji grupy niezmodalizowanej będzie występowało odwołanie do 'pustego' czasownika modalnego.

W wyniku analizy komponentu *have been taken* otrzymujemy zbiór cech gramatycznych (S1),

¹⁵ Większość angielskich czasowników rozróżnia jedynie trzecią osobę liczby pojedynczej.

MAIN_VERB:	take
PASSIVE:	YES
PERFECT:	YES
CONTINUOUS:	NOT
PAST:	NOT
NEGATION:	NOT
MODALITY:	(S1)

natomiast komponent *have been sunk* reprezentuje zbiór (S2):

MAIN_VERB:	sink
PASSIVE:	YES
PERFECT:	YES
CONTINUOUS:	NOT
PAST:	NOT
NEGATION:	NOT
MODALITY:	(S2)

Porównując zbiory (S1) i (S2) można zauważyć, że oba semantycznie różne, lecz syntaktycznie jednakowe komponenty opisuje ten sam zbiór binarnych cech gramatycznych (W1):

PASSIVE:	YES
PERFECT:	YES
CONTINUOUS:	NOT
PAST:	NOT
NEGATION:	NOT (W1)

Formalną reprezentację komponentu o strukturze grupy werbalnej powinien więc tworzyć czasownik główny komponentu w formie podstawowej połączony z czasownikiem modalnym pewną relacją definiującą zbiór binarnych cech gramatycznych komponentu¹⁶.

Powyższe spostrzeżenia stanowią podstawę do zdefiniowania formalnej reprezentacji komponentu o strukturze grupy werbalnej.

3.1.5. Formalna reprezentacja komponentu

Jeżeli przyjmujemy, że podstawowymi elementami bazy wiedzy NLP systemu są słowniki symboli i wyrażenia symboliczne [Kazimierzczak, 1978], to odpowiednią formą reprezentacji elementu bazy wiedzy opisującego komponent o strukturze grupy werbalnej osobowej jest wyrażenie:

¹⁶ W pracy [Unold, 1993] przedstawiono jeszcze inne rozwiązanie problemu.

$$G^+(k_j) = {}^0(d_p^1(z_k k_i d_m^2(z_3 k_i d_p)^2)^1)^0 \quad (11)$$

gdzie

k_i - symbol słownikowy komponentu (struktura słownika K , którego elementami są symbole k_i , jest podana w następnym rozdziale),

d_p - symbol słownikowy formy podstawowej czasownika głównego komponentu (struktura słownika D' , którego elementami są symbole d_p i d_m , jest podana w następnym rozdziale),

d_m - symbol słownikowy czasownika modalnego; w przypadku komponentu niezmodyfikowanego symbol d_0 'pustego' czasownika modalnego,

z_k - relacja definiująca wektor pięciu binarnych cech gramatycznych;

$z_k \in (\text{PASSIVE, PERFECT, CONTINUOUS, PAST, NEGATION})$,

przykładowo $z_{11} = (\text{YES, YES, NOT, NOT, NOT})$,

z_3 - relacja oznaczająca sprzężenie zwrotne.

3.2. Transformacja wiedzy wyrażonej w zdaniu języka naturalnego na postać symboliczną

Proces transformacji wiedzy wyrażonej w zdaniu języka naturalnego (angielskiego) na postać formalną (symboliczną) prześledzimy na przykładzie zdania

Chocolate is poured into moulds at a carefully regulated temperature. (12)

Zakładamy, że dokonujemy początkowej syntezy KB, tzn. iż KB związana z dziedziną systemu jest pusta. W tym przypadku użytkownik systemu oprócz zdania (12) musi podać ciąg symboli

$w_1 w_5 w_5 w_7 w_1 w_7 w_3 w_6 w_4 w_1 w_0$ (13)

reprezentujący kategorie leksykalne poszczególnych słów wchodzących w skład zdania. Każde słowo zdania (12) jest kodowane symbolem d_i ($i=1,2,\dots$), a następnie słowa wraz z odpowiadającymi im symbolami d_i wpisywane są do słownika D' o strukturze przedstawionej na rys. 23. Po zakończonej translacji słów zdania (12) na odpowiadające im symbole d_i powstaje ciąg symboli typu d

$d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8 d_9 d_{10}$ (14)

W momencie powstania ciągu symboli typu d (14) zdanie (12) może zostać wymazane z pamięci komputera.

W przypadku, gdy zdanie zawiera czasownik nieregularny i czasownik ten nie znajduje się w słowniku D' , razem ze zdaniem należy podać wszystkie formy czasownika: bezokolicznik (ang. infinitive), formę past i formę past participle. Dla uproszczenia rozważań zakładamy, że istnieje moduł automatycznego tworzenia wyrazów pochodnych, np. formy $-s$, formy $-ing$, formy $-ed$ czasowników.

słowo	d_i	adres $d_i^{k(}$ w K_{ext}^+
chocolate	d_1	v_1
is	d_2	
poured	d_3	
into	d_4	v_2
moulds	d_5	v_3
at	d_6	v_4
a	d_7	v_5
carefully	d_8	v_6
regulated	d_9	v_7
temperature	d_{10}	v_8
pour	d_{11}	v_9
...	...	...

Rys. 23. Struktura słownika D'

słowo	d_i	w_t
a	d_7	w_3
at	d_6	w_7
carefully	d_8	w_6
chocolate	d_1	w_1
into	d_4	w_7
is	d_2	w_5
moulds	d_5	w_1
pour	d_{11}	w_5
poured	d_3	w_5
regulated	d_9	w_4
temperature	d_{10}	w_1
...	...	...

Rys. 24. Struktura słownika D''

Trzecia kolumna słownika jest adresem $d_i^{k(}$ w wyrażeniu K_{ext}^+ , które zostanie zdefiniowane w dalszej części rozdziału. Jednocześnie ze słownikiem D' powstaje słownik D'' , który w kolumnie trzeciej zawiera symbole kategorii leksykalnych w_t słów uporządkowanych alfabetycznie. Strukturę słownika D'' przedstawiono na rys. 24.

Sekwencja (13) (ciąg (w)) stanowi słowo wejściowe dla automatu $PDAMS$. Automat $PDAMS$ zgodnie z algorytmem działania zilustrowanym przykładem w rozdziale 2.3.6. zapisuje na stosie S_6 następujący ciąg:

$$z_0 6 l_1(w_1) l_2(w_5 w_5) p l_5(w_7 l_1(w_1)) p p l_5(w_7 l_1(w_3 l_3(l_4(w_6) w_4) w_1)) \quad (15)$$

Zgodnie z uwagami poczynionymi wcześniej, na stosie S_6 znajdują się struktury kolejno utworzonych podciągów z ciągu (w), czyli słowa wejściowego. Każdy podciąg (w) $_i$ jest otoczony przez nawiasy okrągłe, stojący przed parą odpowiadających sobie nawiasów symbol l_j określa strukturę syntaktyczną podciągu.

Zawartość stosu S_6 (ciągu (15)) stanowi daną wejściową dla algorytmu zakodowanego w postaci funkcji *tworz_kss* (por. rozdział 2.4.), która wyodrębnia poszczególne komponenty strukturalno-semantyczne. Wynikiem działania funkcji *tworz_kss* są następujące ciągi

(e)	$\leftrightarrow k_1' k_2' k_3' k_5'$
$k_1' (l_1)$	$\leftrightarrow w_1$
$k_2' (l_2)$	$\leftrightarrow w_5 w_5$
$k_3' (l_5)$	$\leftrightarrow w_7 k_4'$
$k_4' (l_1)$	$\leftrightarrow w_1$
$k_5' (l_5)$	$\leftrightarrow w_7 k_6'$
$k_6' (l_1)$	$\leftrightarrow w_3 k_7' w_1$
$k_7' (l_3)$	$\leftrightarrow k_8' w_4$
$k_8' (l_4)$	$\leftrightarrow w_6$

gdzie symbol k_i' ($i=1,\dots,8$) jest oznaczeniem komponentu strukturalno-semantycznego. Przykładowo, w komponencie k_6' o kategorii syntaktycznej grupy nominalnej (przyporządkowany symbol l_1) można wyróżnić podciąg symboli w_1 , który ma odrębną strukturę syntaktyczną i jest oznaczany symbolem k_7' .

Ciąg

$$(e) \leftrightarrow k_1' k_2' k_3' k_4' \quad (16)$$

określa dekompozycję (i jednocześnie strukturę syntaktyczną) ciągu (13), a w konsekwencji zdania (12).

Dzięki jednoznaczному przyporządkowaniu symboli z ciągu (13) do symboli w ciągu (14) można w miejsce symboli typu w_t wpisać odpowiadające im słowa reprezentowane przez symbole d_j . W efekcie otrzymujemy następujące ciągi:

(e)	$\leftrightarrow k_1' k_2' k_3' k_5'$
$k_1' (l_1)$	$\leftrightarrow d_1$
$k_2' (l_2)$	$\leftrightarrow d_2 d_3$
$k_3' (l_5)$	$\leftrightarrow d_4 k_4'$
$k_4' (l_1)$	$\leftrightarrow d_5$
$k_5' (l_5)$	$\leftrightarrow d_6 k_6'$
$k_6' (l_1)$	$\leftrightarrow d_7 k_7' d_{10}$
$k_7' (l_3)$	$\leftrightarrow k_8' d_9$
$k_8' (l_4)$	$\leftrightarrow d_8$

W następnym kroku transformacji wiedzy na postać symboliczną następuje proces analizy gramatycznej komponentu o przyporządkowanym symbolu l_2 , czyli komponentu k_2' definiującego frazę werbalną zdania. W wyniku analizy gramatycznej otrzymujemy ostateczną strukturę komponentu k_2' . Komponent k_2' składa się z ciągu dwóch symboli

$$k_2' (l_2) \leftrightarrow d_{11} d_0 \quad (17)$$

gdzie d_0 oznacza słowo 'puste' (komponent nie jest zmodalizowany). W procesie transformacji komponentu k_2' na wyrażenie symboliczne, symbole d_{11} i d_0 zostaną połączone relacją z_{17} , która definiuje wektor binarnych cech gramatycznych frazy werbalnej *is poured*.

Równoległe z analizą gramatyczną frazy werbalnej następuje zmiana w definicji tych komponentów strukturalno-semantycznych, które złożone są z innych komponentów (w analizowanym przykładzie są to komponenty k_3' , k_5' , k_6' i k_7'). Celem zmiany definicji komponentów jest wyodrębnienie w nich komponentów lub słów reprezentujących rdzenie semantyczne. Dla komponentu reprezentującego frazę przymiotnikową (o przyporządkowanym symbolu l_3) rdzeniem semantycznym jest przymiotnik, dla komponentu reprezentującego frazę przyimkową (symbol l_5) rdzeniem semantycznym jest komponent określający frazę rzeczownikową, rdzeniem semantycznym w frazie rzeczownikowej jest rzeczownik. Zmiana w definicji komponentu polega na przestawieniu na pierwszą pozycję w ciągu określającym komponent rdzenia semantycznego. W rezultacie redefinicji komponentów otrzymujemy następującą, ostateczną ich postać:

(e)	$\leftrightarrow k_1' k_2' k_3' k_5'$
$k_1' (l_1)$	$\leftrightarrow d_1$
$k_2' (l_2)$	$\leftrightarrow d_{11} d_0$
$k_3' (l_5)$	$\leftrightarrow k_4' d_4$
$k_4' (l_1)$	$\leftrightarrow d_5$
$k_5' (l_5)$	$\leftrightarrow k_6' d_6$
$k_6' (l_1)$	$\leftrightarrow d_{10} d_7 k_7'$
$k_7' (l_3)$	$\leftrightarrow d_9 k_8'$
$k_8' (l_4)$	$\leftrightarrow d_8$

W następnym kroku syntezy bazy wiedzy komponenty strukturalno-semantyczne są transformowane na odpowiednie wyrażenia symboliczne. Wynikiem transformacji komponentów są następujące wyrażenia symboliczne

$$\begin{aligned}
 K^+(k_1) &= {}^0(d_1^1(z_3 k_1 d_1)^1)^0 \\
 K^+(k_2) &= {}^0(d_{11}^1(z_{17} k_2 d_0^2(z_3 k_2 d_{11})^2)^1)^0 \\
 K^+(k_3) &= {}^0(k_4^1(z_4 k_3 d_4^2(z_3 k_3 k_4)^2)^1)^0 \\
 K^+(k_4) &= {}^0(d_5^1(z_3 k_4 d_5)^1)^0 \\
 K^+(k_5) &= {}^0(k_6^1(z_4 k_5 d_6^2(z_3 k_5 k_6)^2)^1)^0 \\
 K^+(k_6) &= {}^0(d_{10}^1(z_5 k_6 d_7^2(z_6 k_6 k_7^3(z_3 k_6 d_{10})^3)^2)^1)^0 \\
 K^+(k_7) &= {}^0(d_9^1(z_7 k_7 k_8^2(z_3 k_7 d_9)^2)^1)^0 \\
 K^+(k_8) &= {}^0(d_8^1(z_3 k_8 d_8)^1)^0
 \end{aligned} \quad (18)$$

gdzie z_3 oznacza połączenie zwrotne pomiędzy ostatnim i pierwszym elementem w komponencie (warto zauważyć, że relacja z_3 wskazuje również na rdzeń semantyczny komponentu), z_4 wskazuje na przyimek we frazie przyimkowej, z_5 określa rodzajnik we frazie rzeczownikowej, z_6 wskazuje na przydawkę premodyfikującą [Smólska, 1976] we frazie rzeczownikowej, a z_7 na przydawkę premodyfikującą we frazie przymiotnikowej.

Wyrażenia $K^+(k_i)$, gdzie $i=1,...,8$ są składane w zbiorcze wyrażenie K^+_{ext} , które reprezentuje zbiór komponentów strukturalno-semantycznych w KB:

$$K^+_{ext} = K^+(k_1) \oplus K^+(k_2) \oplus K^+(k_3) \oplus K^+(k_4) \oplus K^+(k_5) \oplus K^+(k_6) \oplus K^+(k_7) \oplus K^+(k_8) =$$

$$= 0(d_1^1(z_3k_1d_1)^1, d_{11}^1(z_17k_2d_0^2(z_3k_2d_{11})^2)^1, k_4^1(z_4k_3d_4^2(z_3k_3k_4)^2)^1, d_5^1(z_3k_4d_5)^1,$$

$$k_6^1(z_4k_5d_6^2(z_3k_5k_6)^2)^1, d_{10}^1(z_5k_6d_7^2(z_6k_6k_7^3(z_3k_6d_{10})^3)^2)^1, d_9^1(z_7k_7k_8^2(z_3k_7d_9)^2)^1, d_8^1(z_3k_8d_8)^1)^0 \quad (19)$$

komponent k_r	pierwszy $k_i(d_j)$ w k_r	l_t	adres k_r^P (w K^+_{ext}	adres k_r^S (w E^+_{ext}
k_1	d_1	l_1		y_1
k_2	d_{11}	l_2		y_2
k_3	k_4	l_5		y_3
k_4	d_5	l_1	x_1	
k_5	k_6	l_5		y_4
k_6	d_{10}	l_1	x_2	
k_7	d_9	l_3	x_3	
k_8	d_8	l_4	x_4	
...	...	...	...	

Rys. 25. Struktura słownika K

Po utworzeniu wyrażenia K^+_{ext} (18) wypełniana jest trzecia kolumna słownika D', która jest adresem d_i^k (w wyrażeniu K^+_{ext} (zob. rys. 23). W miarę nabywania wiedzy z kolejnych zdań, wyrażenie zbiorcze K^+_{ext} rozszerza się o te wyrażenia symboliczne $K^+(k_i)$, które reprezentują nowe, nieznanne jeszcze komponenty strukturalno-semantyczne.

Aby umożliwić odtworzenie struktury dowolnego komponentu wypełniany jest słownik K, o strukturze przedstawionej na rys. 25. Pierwsza kolumna słownika K zawiera symbole k_r ($r=1,2,...$) komponentów, druga kolumna pierwszy element w ciągu definiującym dany komponent, trzecia kolumna jest wypełniana symbolami l_t ($t=1,2,3,4,5,6$) określającymi strukturę syntaktyczną komponentu, czwarta kolumna to adres k_r^P (w wyrażeniu K^+_{ext} , piąta kolumna jest adresem k_r^S (w wyrażeniu E^+_{ext} , które zostanie zdefiniowane w dalszej części rozdziału.

Jak już wspomniano, ciąg (e)(16) definiuje strukturę syntaktyczną zdania (12). Na podstawie tego ciągu konstruowane jest wyrażenie symboliczne

$$E^+(e_I)=^0(k_1^1(z_1e_1k_2^2(z_1e_1k_3^3(z_1e_1k_5^4(z_9e_1k_1)^4)^3)^2)^1)^0 \quad (20)$$

gdzie symbolem e_i ($i=1,2,\dots$) oznaczone są kolejne różne zdania pojawiające się na wejściu systemu, z_1 określa następstwo występowania komponentów w zdaniu, z_9 oznacza relację zwrotną pomiędzy ostatnim i pierwszym komponentem w zdaniu.

Wyrażenie symboliczne $E^+(e_I)$ (20) jest następnie wcielane do zbiorczego wyrażenia E^+_{ext} reprezentującego zbiór wszystkich różnych zdań, z których wiedza uległa transformacji na elementy KB systemu. Podczas początkowej syntezy KB wyrażenie E^+_{ext} zawiera jedynie definicję zdania (12)

$$E^+_{ext}=E^+(e_I) \quad (21)$$

Po utworzeniu wyrażenia E^+_{ext} (21) wypełniana jest piąta kolumna słownika K, która jest adresem k_I^S (w wyrażeniu E^+_{ext} (zob. rys. 25). W celu ułatwienia rekonstrukcji dowolnego zdania ze zbioru zdań zapamiętanych w KB, tworzony jest słownik E, który zawiera początkowe komponenty poszczególnych zdań. Struktura słownika E została przedstawiona na rys. 26.

e_i	pierwszy k_t w e_i
e_1	k_1
...	...

Rys. 26. Struktura słownika E

Podsumowując, na formalną reprezentację wiedzy związanej z dziedziną systemu w KB systemu z dostępem w języku naturalnym składają się słowniki symboli: D' i D'' , K, E oraz odpowiednie wyrażenia symboliczne K^+_{ext} i E^+_{ext} .

3.3. Synteza wiedzy w bazie wiedzy

Nabywając wiedzę od użytkownika lub od inżyniera wiedzy zawartą w kolejnych zdaniach języka naturalnego, system musi rozpoznać, czy nabywana wiedza lub części tej wiedzy (w postaci komponentów strukturalno-semantycznych) nie są już zawarte w KB. Do KB systemu wcielane są wyłącznie nowe części wiedzy.

Rozważmy drugie zdanie, które jest podawane do systemu

The chocolate must be kept at a low temperature (22)

słowo	d_i	adres $d_i.k(\mathbf{v})$ w K_{ext}^+
...	...	...
pour	d_{11}	v_9
the	d_{12}	v_{10}
must	d_{13}	v_{11}
be	d_{14}	
kept	d_{15}	
low	d_{16}	v_{12}
keep	d_{17}	v_{13}
...	...	...

Rys. 27. Struktura słownika D'

Nowym słowom występującym w zdaniu (22) przypisywane są kolejne symbole d_i . Słowa *the*, *must*, *be*, *kept* i *low* otrzymują odpowiednio symbole d_{12} , d_{13} , d_{14} , d_{15} i d_{16} . Przed wpisaniem nowych słów do słowników D' i D'' system pyta użytkownika o kategorie leksykalne tych słów. Po uzyskaniu odpowiedzi rozszerzany jest słownik D' (jednocześnie z porządkowanym alfabetycznie słownikiem D'') o nowe informacje. Na rys. 27 został przedstawiony fragment słownika D' dotyczący nowo nabytych słów ze zdania (22).

W wyniku translacji słów zdania (22) na odpowiadające im symbole d_i powstaje ciąg symboli typu d

$$d_{12}d_{13}d_{14}d_{15}d_{16}d_{17}d_{10} \quad (23)$$

Na podstawie słownika D'' i ciągu d (23) tworzony jest ciąg symboli (w)

$$w_3w_1w_5w_5w_5w_7w_3w_4w_1w_0 \quad (24)$$

Ciągi (23) i (24) stanowią podstawę do dalszej syntezy wiedzy wyrażonej zdaniem (22).

Razem ze zdaniem należy również podać trzy formy czasownika nieregularnego *keep* w formie ciągu (*keep, kept, kept*).

Na podstawie ciągu (24), który jest słowem wejściowym dla automatu *PDAMS*, automat ten generuje na stosie S_6 sekwencję

$$z_{06}p_{l1}(w_3w_1)l_2(w_5w_5w_5)p_{l5}(w_7l_1(w_3l_3(w_4)w_1)) \quad (25)$$

Na bazie sekwencji (25) oraz ciągu symboli typu *d* (23), system produkuje następujące ciągi

(e)	$\leftrightarrow k_1' k_2' k_3'$
$k_1'(l_1)$	$\leftrightarrow d_{12} d_1$
$k_2'(l_2)$	$\leftrightarrow d_{13} d_{14} d_{15}$
$k_3'(l_5)$	$\leftrightarrow d_6 k_4'$
$k_4'(l_1)$	$\leftrightarrow d_7 k_5' d_{10}$
$k_5'(l_3)$	$\leftrightarrow d_{16}$

Symbol k_i' ($i=0,1,\dots,5$) oznacza komponent strukturalno-semantyczny, ciąg (e) określa strukturę składniową zdania (22).

Rezultatem analizy gramatycznej frazy werbalnej, którą reprezentuje komponent k_2' , jest następująca definicja komponentu k_2'

$$k_2'(l_2) \leftrightarrow d_{17} d_{13} \quad (26)$$

(e)	$\leftrightarrow k_1' k_2' k_3'$
$k_1'(l_1)$	$\leftrightarrow d_1 d_{12}$
$k_2'(l_2)$	$\leftrightarrow d_{17} d_{13}$
$k_3'(l_5)$	$\leftrightarrow k_4' d_6$
$k_4'(l_1)$	$\leftrightarrow d_{10} d_7 k_5'$
$k_5'(l_3)$	$\leftrightarrow d_{16}$

(27)

Podczas transformacji komponentu k_2' na wyrażenie symboliczne, symbole d_{17} i d_{13} połączy relacja z_{18} , która definiuje wektor binarnych cech gramatycznych frazy *must be kept*. Po przestawieniu na pierwszą pozycję rdzeni semantycznych w komponentach złożonych, zbiór komponentów wyróżnionych w zdaniu (22) ma postać zestawu (27).

Jak już wspomniano na wstępie tego rozdziału, podczas syntezy nowych zdań system musi stwierdzić, czy dana częśćka wiedzy (komponent strukturalno-semantyczny)

nie jest już reprezentowana w wyrażeniu K^+_{ext} (19). W pierwszym etapie szukania nowych komponentów stosowana jest reguła R1.

Reguła R1

Komponent, który składa się z nowego słowa (czyli słowa dopiero co wpisanego do słowników D' i D'') reprezentuje nowy komponent. Komponent ten otrzymuje symboliczną nazwę k_i , gdzie indeks i jest kolejnym numerem indeksu zbioru komponentów w KB.

Według reguły R1 w zestawie komponentów (27) wyodrębnione będą następujące nowe komponenty:

$$k_9 = k_1', \quad k_{10} = k_2', \quad k_{11} = k_5' \quad (28)$$

Druga reguła, reguła R2, stosowana podczas szukania nowych komponentów w analizowanym zdaniu ma postać rekurencyjną:

Reguła R2

Komponent, który składa się przynajmniej z jednego komponentu oznaczonego nowym indeksem stanowi nowy komponent. Otrzymuje on symboliczną nazwę k_i z kolejnym numerem indeksu i .

Reguła R2 znajduje w zestawie (27) następujące nowe komponenty:

$$k_{12} = k_4', \quad k_{13} = k_3' \quad (29)$$

Efektorem zastosowania reguł R1 i R2 do zestawu komponentów (27) jest następujący zbiór komponentów wyróżnionych w zdaniu (22)

(e)	$\leftrightarrow k_9 k_{10} k_{13}$
$k_9 (l_1)$	$\leftrightarrow d_1 d_{12}$
$k_{10} (l_2)$	$\leftrightarrow d_{17} d_{13}$
$k_{13} (l_5)$	$\leftrightarrow k_{12} d_6$
$k_{12} (l_1)$	$\leftrightarrow d_{10} d_7 k_{11}$
$k_{11} (l_3)$	$\leftrightarrow d_{16}$

(30)

Jeżeli w otrzymanym zestawie (30) wystąpiłyby komponenty bez przydzielonego przez reguły R1 i R2 symbolu k_i (tzn. komponenty oznaczone symbolem k_i'), to w takim wypadku system musi sprawdzić na podstawie wyrażenia K^+_{ext} , czy dana sekwencja k_i' nie

jest już reprezentowana w KB. Jeżeli w trakcie procesu poszukiwania system stwierdzi, że dana sekwencja nie ma swojej reprezentacji w KB, to przydziela sekwencji symbol k_i z kolejnym numerem indeksu i .

W kolejnym kroku syntezy bazy wiedzy komponenty transformowane są na następujące wyrażenia symboliczne:

$$\begin{aligned}
 K^+(k_9) &= (d_1^1(z_5k_9d_{12}^2(z_3k_9d_1)^2)^1)^0 \\
 K^+(k_{10}) &= (d_{17}^1(z_{18}k_{10}d_{13}^2(z_3k_{10}d_{17})^2)^1)^0 \\
 K^+(k_{11}) &= (d_{16}^1(z_3k_{11}d_{16})^1)^0 \\
 K^+(k_{12}) &= (d_{10}^1(z_5k_{12}d_7^2(z_6k_{12}k_{11}^3(z_3k_{12}d_{10})^3)^2)^1)^0 \\
 K^+(k_{13}) &= (k_{12}^1(z_4k_{13}d_6^2(z_3k_{13}k_{12})^2)^1)^0
 \end{aligned} \tag{31}$$

gdzie z_3, z_4, z_5, z_6 zostały zdefiniowane w rozdziale poprzednim.

Wyrażenie zbiorcze K^+_{ext} po wcieleniu wyrażeń (31) przyjmuje następującą postać:

$$\begin{aligned}
 K^+_{ext} = & (d_1^1(z_3k_1d_1, z_5k_9d_{12}^2(z_3k_9d_1)^2)^1)^1, & d_{11}^1(z_{17}k_2d_0^2(z_3k_2d_{11})^2)^1, \\
 & k_4^1(z_4k_3d_4^2(z_3k_3k_4)^2)^1, & d_5^1(z_3k_4d_5)^1, & k_6^1(z_4k_5d_6^2(z_3k_5k_6, z_3k_{13}k_{12})^2)^1, \\
 & d_{10}^1(z_5k_{12}d_7, z_5k_6d_7^2(z_6k_{12}k_{11}, z_6k_6k_7^3(z_3k_6d_{10})^3)^2)^1, & d_9^1(z_7k_7k_8^2(z_3k_7d_9)^2)^1, \\
 & d_8^1(z_3k_8d_8)^1, & d_{16}^1(z_3k_{11}d_{16})^1, & d_{17}^1(z_{18}k_{10}d_{13}^2(z_3k_{10}d_{17})^2)^1, & k_{11}^1(z_3k_{12}d_{10})^1, \\
 & k_{12}^1(z_4k_{13}d_6)^1)^0
 \end{aligned} \tag{32}$$

komponent k_r	pierwszy $k_i(d_j)$ w k_r	l_t	adres k_r^P (w K^+_{ext}	adres k_r^S (w E^+_{ext}
...	...	...	...	...
k_8	d_8	l_4	x_4	
k_9	d_1	l_1		y_5
k_{10}	d_{17}	l_2		y_6
k_{11}	d_{16}	l_3	x_5	
k_{12}	d_{10}	l_1	x_6	
k_{13}	k_{12}	l_5		y_7
...	...	...	...	...

Rys. 28. Struktura słownika K

Po otrzymaniu wyrażenia K^+_{ext} (32) system uzupełnia słownik K o nowe komponenty strukturalno-semantyczne k_i ($i=9, \dots, 13$) co ilustruje rys. 28 oraz wypełnia trzecią kolumnę słownika D'.

Ciąg (e) z zestawu komponentów (30) reprezentuje strukturę syntaktyczną zdania (22). Ponieważ w skład ciągu wchodzi nowo nabyte komponenty (k_9 , k_{10} , k_{13}), ciąg (e) definiuje strukturę zdania, które jest różne od zdań zapamiętanych w KB.

Gdyby nowe zdanie było reprezentowane przez ciąg (e), w skład którego nie wchodziłby żaden nowy komponent, to system posługując się wyrażeniem E^+_{ext} i słownikiem K musiałby sprawdzić, czy sekwencja (e) nie jest już reprezentowana w KB. Odpowiedź twierdząca świadczyłaby o tym, że zdanie ma już swoją reprezentację w KB systemu.

Na podstawie struktury ciągu (e) z zestawu (30) budowane jest wyrażenie

$$E^+(e_2)=^0(k_9^1(z_1e_2k_{10}^2(z_1e_2k_{13}^3(z_9e_2k_9)^3)^2)^1)^0 \quad (33)$$

w którym symbolem e_2 oznaczono zdanie (22).

Wyrażenie $E^+(e_2)$ (33) zostaje wcielone do zbiorczego wyrażenia E^+_{ext} , które otrzymuje postać

$$E^+_{ext}=^0(k_1^1(z_1e_1k_2^2(z_1e_1k_3^3(z_1e_1k_5^4(z_9e_1k_1)^4)^3)^2)^1, \quad k_9^1(z_1e_2k_{10}^2(z_1e_2k_{13}^3(z_9e_2k_9)^3)^2)^1)^0 \quad (34)$$

Po uzyskaniu wyrażenia E^+_{ext} system uzupełnia słownik E o nowy symbol e_2 i rozpoczynający zdanie oznaczone e_2 komponent k_9 oraz wpisuje w piątej kolumnie słownika K adresy termów k_9^1 , k_{10}^2 i k_{13}^3 .

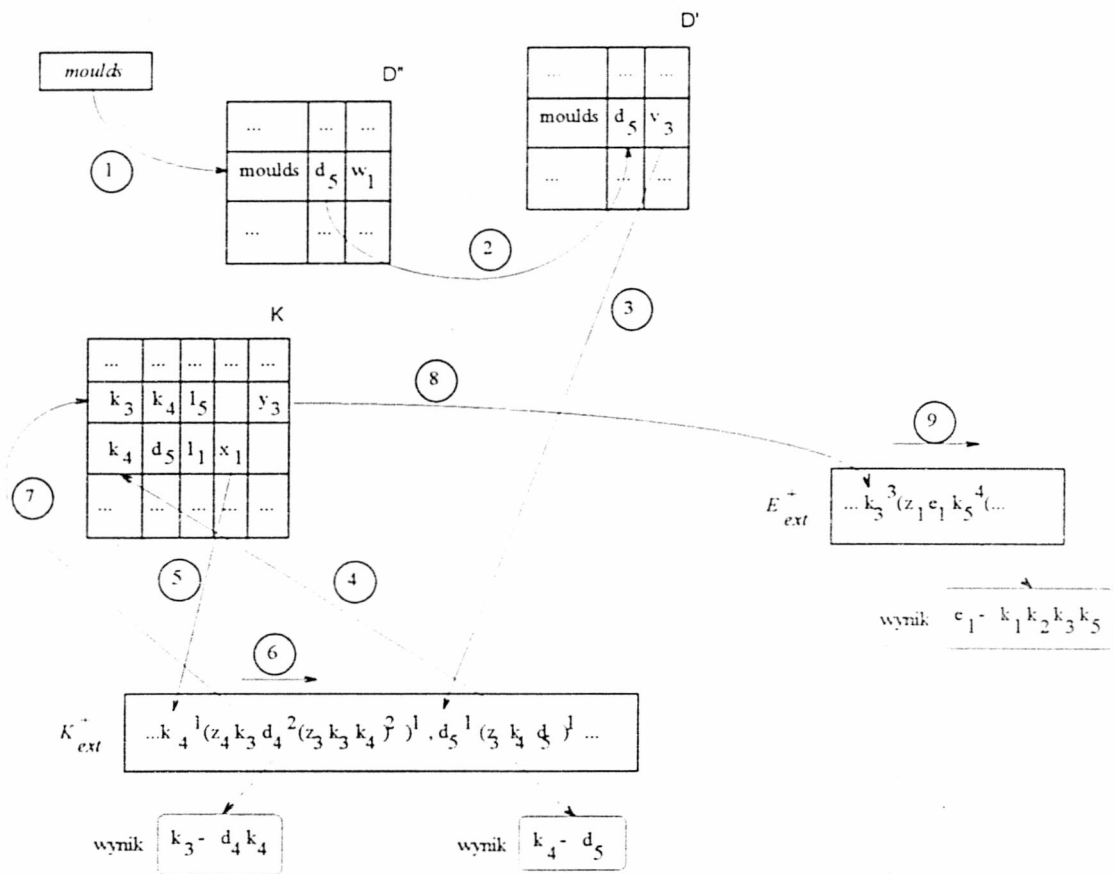
3.4. Odtwarzanie wiedzy z bazy wiedzy

Przedstawiony w rozdziałach poprzednich formalny system reprezentacji wiedzy wyrażonej w zdaniach języka naturalnego umożliwia rekonstrukcję zdań z KB systemu w takiej formie, w jakiej zostały do systemu podane. Proces reprodukcji wiedzy z bazy wiedzy zostanie przedstawiony na przykładzie odtwarzania z KB zdania zawierającego słowo *moulds*. Zakładamy, że użytkownik zadał pytanie, w którym słowo to występuje. Niektóre akcje procesu rekonstrukcji zostały przedstawione na rys. 29. Akcje oznaczono kolejnymi cyframi w okręgach.

W pierwszej kolejności słowo *moulds* poszukiwane jest w słowniku D" (słownik ten zawiera słowa uporządkowane alfabetycznie). W KB słowo to jest reprezentowane przez symbol d_5 (akcja 1). Symbol d_5 jest następnie poszukiwany w słowniku D' (słownik ten zawiera słowa uporządkowane według wzrastających indeksów symboli d_j) - akcja 2. Na

podstawie adresu v_3 pobranego z trzeciej kolumny słownika D' w wyrażeniu K^+_{ext} znajdujący jest term $d_5^1(z_3k_4d_5)^1$ - akcja 3. Term ten oznacza, że słowo *moulds* tworzy komponent strukturalno-semantyczny k_4 . Symbol k_4 jest poszukiwany w słowniku K (akcja 4). W czwartej kolumnie słownika K znajduje się adres termu k_4^T (w wyrażeniu K^+_{ext}). Na podstawie tego adresu znajdujący jest term k_4^1 (akcja 5). W następnej kolejności (akcja 6) poszukiwane jest w wyrażeniu K^+_{ext} powiązanie elementu k_4 z innymi elementami. Wynik operacji poszukiwania można zapisać w formie wyrażenia symbolicznego

$$^0(k_4^1(z_4k_3d_4^2(z_3k_3k_4)^2)^1)^0 \quad (35)$$



Rys. 29. Fragment przykładowej rekonstrukcji wiedzy z bazy wiedzy

Wyrażenie (35) definiuje komponent k_3 , który składa się z elementów k_4 i d_4 (komponent k_4 jest rdzeniem semantycznym komponentu k_3 , ponieważ w wyrażeniu definiującym k_3 wskazywany jest przez relację z_3). O ostatecznym porządku elementów w komponencie k_3 decyduje relacja z_4 wskazująca na pierwszy element (przyimek o symbolu d_4 w grupie przyimkowej). Tak więc komponent k_3 definiuje sekwencja d_4d_5 , którą uzyskujemy po zastąpieniu komponentu k_4 symbolem d_5 . Następnie (akcja 7) w słowniku K poszukiwany jest symbol k_3 . Ponieważ pole adresu k_3^T (w słowniku K jest puste, możemy wnioskować,

że komponent k_3 nie wchodzi w skład żadnego komponentu, Na podstawie adresu y_3 pobranego z piątej kolumny słownika K znajdujemy w wyrażeniu E^+_{ext} term k_3^3 (akcja 8). Następnie poszukiwane jest w wyrażeniu E^+_{ext} powiązanie elementu k_3 z innymi elementami k_i - akcja 9. Rezultatem poszukiwania jest wyrażenie symboliczne

$$k_3^1(z_1e_1k_5^2(z_9e_1k_1^3(z_1e_1k_2^4(z_1e_1k_3^4)^3)^2)^1) \quad (36)$$

Wyrażenie (36) definiuje zdanie oznaczane symbolem e_1 . Porządek komponentów w zdaniu określa relacja z_9 , która wskazuje na pierwszy komponent w zdaniu. Tak więc zdanie e_1 tworzy następujący ciąg komponentów:

$$k_1 \ k_2 \ k_3 \ k_5 \quad (37)$$

Po otrzymaniu sekwencji (37), która reprezentuje zdanie e_1 , należy odtworzyć wchodzące w skład zdania komponenty k_1 , k_2 i k_5 (komponent k_3 został już zrekonstruowany w postaci ciągu symboli d_4d_5).

Na podstawie słownika K uzyskujemy informacje, że pierwszym elementem komponentu k_1 jest element d_1 . Po znalezieniu symbolu d_1 w słowniku D' przechodzimy do miejsca w wyrażeniu K^+_{ext} wskazywanym przez adres z trzeciej kolumny słownika D' . Pod adresem tym znajduje się term $d_1^1(z_3k_1d_1)^1$, który oznacza, że komponent k_1 składa się jedynie z elementu d_1 .

W kolejnym kroku odtwarzania wiedzy w słowniku K szukany jest symbol k_2 . Następnie poszukiwany jest w słowniku D' symbol d_{11} stanowiący pierwszy element komponentu k_2 o strukturze grupy werbalnej (symbol l_2). Na podstawie adresu z trzeciej kolumny słownika D' znajdujemy w wyrażeniu K^+_{ext} term d_{11}^1 . W następnej kolejności poszukiwane jest w wyrażeniu K^+_{ext} powiązanie elementu d_{11} z innymi elementami d_i . Wynik operacji poszukiwania można zapisać w formie wyrażenia symbolicznego

$$^0(d_{11}^1(z_{17}k_2d_0^2(z_3k_2d_{11})^2)^1)^0 \quad (38)$$

Ponieważ komponent k_2 ma strukturę grupy werbalnej osobowej, to po otrzymaniu czasownika głównego komponentu (o symbolu d_{11}), czasownika modalnego (w przypadku komponentu k_2 jest to czasownik 'pusty' d_0) oraz relacji z_{17} definiującej wektor binarnych cech gramatycznych komponentu następuje proces syntezy gramatycznej. W wyniku syntezy gramatycznej otrzymujemy ciąg słów tworzących komponent k_2 . Ostatnim etapem rekonstrukcji komponentu o strukturze grupy werbalnej jest ustalenie formy osobowej pierwszego czasownika w komponencie.

W kolejnym kroku procesu rekonstrukcji wiedzy odtwarzany jest komponent k_5 . Pierwszym elementem komponentu k_5 jest komponent k_6 . Po znalezieniu w słowniku K symbolu k_6 pobierany jest z czwartej kolumny słownika adres termu k_6P (w wyrażeniu K^+_{ext}). Od wskazanego przez adres miejsca poszukiwane jest powiązanie elementu k_6 z innymi elementami w komponencie k_5 . Rezultatem poszukiwania jest wyrażenie symboliczne

$$^0(k_6^1(z_4k_5d_6^2(z_3k_5k_6)^2)^1)^0 \quad (39)$$

O porządku elementów k_6 i d_6 w odtwarzanym komponencie k_5 decyduje relacja z_4 , która wskazuje pierwszy element (przymek w grupie przymkowej). Stąd komponent k_5 tworzy sekwencja d_6k_6 . W następnym kroku odtwarzany jest komponent k_6 . Pierwszym elementem komponentu k_6 jest symbol d_{10} . Na podstawie słownika D' znajdowany jest adres termu d_{10}^k (w wyrażeniu K^+_{ext}). Od tego termu poszukiwane jest w wyrażeniu K^+_{ext} powiązanie elementu d_{10} z innymi elementami w komponencie k_6 . Wynik operacji poszukiwania można zapisać w formie wyrażenia symbolicznego

$$^0(d_{10}^1(z_5k_6d_7^2(z_6k_6k_7^3(z_3k_6d_{10})^3)^2)^1)^0 \quad (40)$$

O kolejności elementów d_{10} , d_7 i k_7 w odtwarzanym komponencie k_6 decydują relacje z_5 i z_6 . Relacja z_5 wskazuje na pierwszy element w ciągu (rodzajnik w grupie rzeczownikowej), relacja z_6 na element drugi (przydawkę premodyfikującą w grupie rzeczownikowej). Kolejność elementów w komponencie k_6 jest więc następująca: $d_7k_7d_{10}$. W kolejnym kroku rekonstruowany jest komponent k_7 . Pierwszym elementem komponentu jest element d_9 . Na podstawie słownika D' uzyskiwany jest adres termu d_9^l (w wyrażeniu K^+_{ext}). Od termu d_9^l poszukiwane jest powiązanie elementu d_9 z innymi elementami w komponencie k_7 . Wynikiem poszukiwania jest wyrażenie symboliczne

$$^0(d_9^1(z_7k_7k_8^2(z_3k_7d_9)^2)^1)^0 \quad (41)$$

Porządek elementów d_9 i k_8 w komponencie k_7 określa relacja z_7 , która wskazuje na pierwszy element (przydawkę premodyfikującą w grupie przymiotnikowej). Komponent k_7 tworzy sekwencja: k_8d_9 . W następnym kroku odtwarzany jest komponent k_8 . Na podstawie słownika K określany jest pierwszy element komponentu k_8 , którym jest element d_8 . Ze słownika D' pobierany jest adres termu k_8P (w wyrażeniu K^+_{ext}). Adres ten wskazuje na term $d_8^l(z_3k_8d_8)^l$, który oznacza, że komponent k_8 składa się z elementu d_8 .

Podstawiając kolejno w miejsce symboli k_j uzyskane ciągi symboli d_j , zaczynając od

podstawienia w miejsce k_8 symbolu d_8 , otrzymujemy definicję komponentu k_5 . Komponent k_5 tworzy następująca sekwencja symboli d_j : $d_6d_7d_8d_9d_{10}$.

W ostatnim kroku rekonstrukcji zdania e_1 sekwencje symboli d_j określające komponenty k_1 , k_3 i k_5 transformowane są przy użyciu słownika D' w słowa języka angielskiego (komponent k_2 został przekształcony na sekwencję słów poprzez proces syntezy gramatycznej).

4. PROPOZYCJA REPREZENTACJI WIEDZY DLA POTRZEB ANALIZY SEMANTYCZNEJ

Przedstawiony w rozdziale poprzednim formalny system reprezentacji wiedzy wyrażonej w zdaniach języka naturalnego ułatwia przeprowadzenie analizy semantycznej wypowiedzi. Bowiem już w procesie dekompozycji zdania na elementy bazy wiedzy zostają wyodrębnione w zdaniu jednostki semantyczne, każda o charakterystycznej strukturze składniowej, nazywane w prezentowanej pracy komponentami strukturalno-semantycznymi. Co więcej, w symbolicznym zapisie komponentów w KB wyróżniane są ich rdzenie semantyczne, których wydzielenie upraszcza proces wnioskowania oparty na operacji dopasowywania (ang. pattern matching). Analiza semantyczna zdania polegać będzie na przydzieleniu odpowiednich kategorii semantycznych do komponentów zdania. Skojarzenie komponentu z kategorią semantyczną umożliwi przeprowadzenie na bazie wiedzy wnioskowania proceduralnego.

4.1. Zbiór kategorii semantycznych

Jedną z najczęściej stosowanych metod reprezentacji znaczenia języka naturalnego jest gramatyka przypadków głębokich Fillmore'a [1968]. Pomimo swoistej popularności "case grammar" w badaniach nad automatycznym przetwarzaniem języka naturalnego nie jest ona wolna od krytycznych uwag badaczy, w tym również samego twórcy teorii. Między innymi zarzuca się gramatyce przypadków głębokich, że:

- jest tylko pewnym wariantem zapisu zjawisk językowych znanych również w innych teoriach lingwistycznych,
- nie jest pełnym modelem lingwistycznym,
- nie ma ogólnie przyjętego zbioru przypadków głębokich; co więcej twierdzi się [Winograd, 1983], że nie można uniwersalnie i ostatecznie zdefiniować tego zbioru,
- przypadki głębokie (kategorie semantyczne) jedynie powtarzają część znaczenia predykatu, tzn. nie znając znaczenia czasownika można równie dobrze w miejsce przypadku 'agent' czy 'patient' użyć określenia 'podmiot', 'dopełnienie',
- wyznaczenie kategorii semantycznej jest często utrudnione, choć konieczne

Przykładowo, w zdaniu *John coughs* podmiot zdania *John* może występować zarówno w roli 'agent', jak i 'experiencer'.

Równie znaną teorią reprezentacji znaczenia języka naturalnego co "case grammar", również wykorzystującą pojęcie przypadków (kategorii) wyrażających relację pomiędzy

akcjami i ich obiektami, jest "conceptual dependency" Schank'a [Schank, Colby, 1973], [Schank, 1975]. W systemie uczącym się, nabywającym wiedzę nie tylko o otoczeniu, ale również i o samym języku, zastosowanie teorii CD jako metody reprezentacji wiedzy napotyka poważne trudności. Wynikają one z faktu, że w tej teorii wszystkie przypadki związane z elementarnymi akcjami (ang. primitive acts) są obligatoryjne. Stąd, w reprezentacji opartej na CD mogą wystąpić elementy, których nie ma w strukturze powierzchniowej zdania. Zastosowanie teorii Schank'a wymaga obecności w NLP systemie odpowiednio zorganizowanego, a przede wszystkim zamkniętego modułu wiedzy lingwistycznej.

Interesującą propozycją reprezentacji znaczenia zdania języka naturalnego wydaje się być system Halliday'a, tzw. "systemic grammar" [Halliday, 1967]. W modelu tym gramatyka przypadków głębokich (kategorii semantycznych) stanowi jego integralną część, choć same kategorie nazywane są "transitive roles". Podczas definiowania kategorii semantycznych w gramatyce systemowej wykorzystywany jest jeden z ważniejszych mechanizmów modelu, tzn. proces klasyfikacji (ang. classification). W miejsce jednego zbioru kategorii definiuje się odrębny zbiór dla każdego z kilku podstawowych procesów, jakie mogą występować w zdaniu. Każde zdanie rozpatrywane jest jako pewien proces (lub relacja), z jednym lub więcej niż jednym uczestnikiem oraz ewentualnymi okolicznościami. W systemie Halliday'a wyróżnia się pięć podstawowych procesów. Każdy czasownik jest związany z jakimś typem procesu i zestawem uczestników procesu oraz okoliczności, tworząc tzw. ramę przypadków (ang. case frame). Gramatyka systemowa stanowi więc swoistą wypadkową pomiędzy teorią Fillmore'a a teorią Schank'a.

Przewaga systemu Halliday'a nad gramatyką przypadków głębokich i teorią CD wynika z następujących czynników:

- istnienie różnych zbiorów przypadków dla różnych typów procesów umożliwia semantyczne rozróżnienie ról, jakie pełnią uczestnicy procesu w zależności od typu akcji,
- mniejszy zbiór elementarnych akcji (procesów) w stosunku do teorii CD ułatwia klasyfikację czasownika do typu procesu,
- podczas wyznaczania kategorii biorą udział jedynie obiekty wzięte ze struktury powierzchniowej zdania.

Wyżej wymienione czynniki spowodowały, że za podstawę reprezentacji semantyki zdania języka naturalnego przyjęto zmodyfikowane przez autora niniejszej pracy kategorie semantyczne Halliday'a. Bez zmiany zachowane zostały typy procesów, jakie można rozróżnić w zdaniach:

- proces materialny (ang. material process clauses),
- proces myślowy (ang. mental process clauses),
- proces werbalny (ang. verbal process clauses),

- proces atrybutowy (ang. attributive clauses),
- proces identyfikujący (ang. equative clauses).

Z każdym typem procesu związany jest inny zbiór kategorii semantycznych (przypadków), w których występują uczestnicy procesu oraz niezależny od typu procesu zestaw okoliczności towarzyszących procesowi.

4.1.1. Kategorie semantyczne procesu materialnego

Proces materialny opisuje większość angielskich czasowników. Uczestnicy tego typu procesu mogą występować w następujących kategoriach semantycznych: Agent, Medium, Beneficiary. Kategorie procesu materialnego ilustruje poniższe zdanie

John gave the book to Jack for Susan
 Agent Medium Beneficiary

4.1.2. Kategorie semantyczne procesu myślowego

Proces myślowy reprezentują między innymi następujące czasowniki: *like, please, see, hear, think, know*. W zdaniu, w którym występuje jeden z powyższych czasowników zawsze można wyróżnić komponenty strukturalno-semantyczne o kategoriach semantycznych Sener i Phenomenon. Kategorie procesu myślowego ilustruje następujące zdanie

She liked it
 Sener Phenom.

4.1.3. Kategorie semantyczne procesu werbalnego

Proces werbalny w zdaniu jest opisywany przez wiele czasowników, np.: *say, tell, ask, describe, call*. W zdaniach tego typu wyróżniamy następujące kategorie semantyczne: Sayer, Verbalization, Addressee. Kategorie te ilustruje poniższe zdanie

We told a story to the campers
 Sayer Verbaliz. Addressee

4.1.4. Kategorie semantyczne procesu atrybutowego

W zdaniu wyznaczającym pewne atrybuty obiektów najczęściej pojawiają się czasowniki: *be*, *become*, *stay*. Uczestników tego typu procesu nazywamy: Carrier i Attribute. Kategorie semantyczne procesu przedstawiono na przykładzie poniższego zdania

The experts remained in the dark

Carrier

Attribute

4.1.5. Kategorie semantyczne procesu identyfikującego

W zdaniach, w których następuje identyfikacja obiektu wyróżniamy dwie kategorie semantyczne: Identified i Identifier. Kategorie te ilustruje poniższe zdanie

The one on the left is my mother

Identified

Identifier

4.1.6. Kategorie semantyczne okoliczności towarzyszących procesowi

W zdaniu, niezależnie od typu procesu jaki reprezentuje, mogą występować różne typy okoliczności towarzyszących procesowi. Proponuje się przyjąć następujące kategorie semantyczne dla okoliczności:

- Instrument; odpowiada na pytania *what with?*, *by what means?* (np. *with a stick*),
- Location - rozumiana przestrzennie, czasowo lub pojęciowo; odpowiada na pytania *when?*, *where?*, *when at?*, *by what?*, *in what time?*, *in what?*, *during when?*, *before when?* (np. *at that time*, *at school*),
- Source - rozumiana przestrzennie, czasowo lub pojęciowo; odpowiada na pytania *where from?*, *from what?*, *out of what?* (np. *from the farm*, *out of wood*),
- Goal - rozumiana przestrzennie, czasowo lub pojęciowo; odpowiada na pytania *where to?*, *what end?*, *for what purpose?* (np. *to the lake*),
- Accompaniment; odpowiada na pytania *accompanied by what/who?*, *with what/whom?*, *without what/whom?* (np. *with Jack*, *with a glass*).

4.2. Formalna reprezentacja atrybutów semantycznych elementów bazy wiedzy

Zgodnie z założeniami systemu Halliday'a czasownik pełniący funkcję orzeczenia zasadniczego w analizowanym zdaniu określa jeden z pięciu podstawowych typów

procesów, jakie zdanie może opisywać (proces materialny, myślowy, werbalny, atrybutowy oraz identyfikujący). Od typu procesu zależy jaki zbiór kategorii semantycznych można przypisać do komponentów strukturalno-semantycznych wyróżnionych przez automat *PDAMS* w zdaniu. Aby zaklasyfikowanie zdania do opisywanego przez to zdanie typu procesu mogło nastąpić automatycznie, musi być pamiętana informacja kojarząca czasownik z odpowiednim typem procesu. Jeżeli czasownik główny komponentu o strukturze grupy werbalnej osobowej nie istnieje w KB systemu, to użytkownik (inżynier wiedzy) musi sam dokonać klasyfikacji czasownika. W bazie wiedzy informacja o typie procesu opisywanego przez dany czasownik może być pamiętana w dodatkowej, czwartej kolumnie słownika *D'* w formie symbolu np. s_i ($i=1, \dots, 5$). Interpretacja symbolów s_i jest następująca:

- s_1 - proces materialny,
- s_2 - proces myślowy,
- s_3 - proces werbalny,
- s_4 - proces atrybutowy,
- s_5 - proces identyfikujący.

Przyjmujemy założenie, że kategorie semantyczne komponentów strukturalno-semantycznych nie są pamiętane w KB systemu, lecz są wypracowywane według określonych reguł w trakcie działania procedury wnioskowania.

4.3. Możliwości przeprowadzenia wnioskowania proceduralnego

Omówiona metoda reprezentacji wiedzy umożliwia stworzenie lingwistycznego mechanizmu wnioskującego realizującego wnioskowanie proceduralne oparte na operacji dopasowywania i wykorzystujące skojarzenie komponentów z kategoriami semantycznymi.

Proces wnioskowania jest inicjowany przez pytanie użytkownika dotyczące zawartości bazy wiedzy systemu. Wybór odpowiedniej procedury wnioskującej jest uzależniony od struktury zdania pytającego (czy jest to pytanie ogólne - ang. general question, czy też pytanie szczegółowe - ang. special question) oraz w przypadku pytania szczegółowego od typu wyrazu pytającego. Przedstawienie kompletnego algorytmu wnioskowania wykracza poza charakter niniejszej pracy, a ze względu na zakres problematyki wymaga oddzielnego rozpatrzenia. Dlatego też w tym miejscu zostanie przedstawiony w sposób schematyczny jedynie prosty przykład ilustrujący potencjalne możliwości mechanizmu wnioskującego.

Zakładamy, że w KB reprezentowane są rozpatrzone już wcześniej zdania e_1 i e_2 oraz zdania e_3 i e_4 o następującej treści:

The chocolate represents the sweets. (42)

Sweets are the high calorific groceries. (43)

Pytanie użytkownika ma postać następującego zdania

Does milk chocolate represent the high calorific groceries? (44)

Zauważmy, że na pytanie użytkownika można odpowiedzieć dopiero na podstawie informacji zawartej w dwóch zdaniach: $e_3(42)$ i $e_4(43)$.

W pierwszej kolejności analizie podlega struktura zdania pytającego (44). Jest to pytanie ogólne, na które odpowiedź twierdząca miałaby postać zdania

Milk chocolate represents the high calorific groceries. (45)

Zadaniem procedury wnioskującej jest więc udowodnienie prawdziwości zdania (45).

Ponieważ słowo *milk* nie ma jeszcze swojej reprezentacji w słownikach D' i D'' , to procedura wnioskująca nie sprawdza, czy zdanie e_5 jest już reprezentowane w KB, lecz nadaje mu symboliczną nazwę e_5 . Użytkownik systemu musi podać kategorię leksykalną słowa *milk* (symbol w_j). Słowa zdania e_5 zostają zakodowane symbolami d_i , a całe zdanie zostaje zdekomponowane przez automat *PDAMS* na komponenty strukturalno-semantyczne.

Następnie procedura wnioskująca bada jaki typ procesu opisuje zdanie e_5 . Z czasownikiem *represent* skojarzony jest w słowniku D' symbol s_5 , który reprezentuje proces identyfikujący. Na podstawie tej informacji komponent *milk chocolate* otrzymuje kategorię semantyczną Identified, komponent *the high calorific groceries* kategorię Identifier.

W kolejnym kroku procedura wnioskująca szuka w KB zdania, w którym występuje komponent *the high calorific groceries*. Na podstawie słownika K i wyrażenia E^+_{ext} zostaje znalezione zdanie e_4 (43).

Zdanie e_4 opisuje również proces identyfikujący (z czasownikiem *be* skojarzony jest w słowniku D' symbol s_5), w którym komponent *sweets* ma kategorię Identified, natomiast komponent *the high calorific groceries* kategorię Identifier. Ponieważ oba zdania $e_5(45)$ i $e_4(43)$ opisują ten sam proces identyfikujący, w którym komponent *the high calorific groceries* pełni tę samą funkcję semantyczną Identifier, to wystarczy udowodnić

równoważność semantyczną komponentów *milk chocolate* i *sweets*, aby wykazać prawdziwość zdania e_5 .

W następnym kroku procedura wnioskująca poszukuje w KB zdania (za wyjątkiem analizowanego już zdania e_4), w którym występuje komponent *sweets* lub komponent o takim rdzeniu semantycznym. Na podstawie słownika K i wyrażenia E^+_{ext} zostaje znalezione zdanie $e_3(42)$, w którym rdzeniem semantycznym komponentu *the sweets* jest rzeczownik *sweets*.

Zdanie e_3 opisuje proces identyfikujący, w którym komponent o rdzeniu semantycznym *sweets* ma kategorię Identified, natomiast komponent *the chocolate* kategorię Identifier. Ponieważ rdzenie semantyczne komponentów *the chocolate* i *milk chocolate* są identyczne (rzeczownik *chocolate*) można przyjąć, że komponenty *milk chocolate* i *sweets* są równoważne semantycznie. Ponieważ oba komponenty pełnią identyczną funkcję semantyczną (kategoria Identified) w procesie identyfikującym komponent *the high calorific groceries*, stąd na podstawie prawdziwości zdania $e_4(43)$ wywodzona jest prawdziwość zdania $e_5(45)$. Odpowiedź na pytanie użytkownika będzie więc twierdząca.

5. PODSUMOWANIE

W pracy został przedstawiony formalny model reprezentacji wiedzy wyrażonej w zdaniach języka naturalnego (angielskiego) w bazie wiedzy systemu z dostępem w języku naturalnym. Model ów bazuje na koncepcji zaproponowanej i rozwijanej przez J. Kazimierczaka, która w pracy została poddana istotnym modyfikacjom. Wprowadzone zmiany spowodowały m.in. rozszerzenie występującego w modelu formalizmu wyrażen symbolicznych.

Wprowadzono pojęcie komponentu strukturalno-semantycznego i zdefiniowano model automatu *PDAMS*, za pomocą którego jest dokonywana dekompozycja zdania pojedynczego na komponenty strukturalno-semantyczne. W jednym z załączników pracy jest zamieszczony tabulogram programu stanowiącego realizację (a jednocześnie weryfikację) programową automatu.

Poszukiwania jak najefektywniejszej reprezentacji wiedzy podawanej w formie zdań języka naturalnego zaowocowały koncepcją komputerowej reprezentacji komponentu o strukturze syntaktycznej grupy werbalnej osobowej.

Prezentowany formalny system reprezentacji wiedzy umożliwia analizę semantyczną wypowiedzi, która opiera się na zmodyfikowanym zbiorze kategorii semantycznych oraz mechanizmie klasyfikacji gramatyki systemowej.

W pracy zademonstrowano również w formie przykładu możliwość przeprowadzania wnioskowania proceduralnego w bazie wiedzy systemu wykorzystującego prezentowany formalny system reprezentacji wiedzy.

Przedstawiona w pracy koncepcja organizacji bazy wiedzy NLP systemu stanowi podstawę do prowadzenia dalszych prac naukowych z zakresu przetwarzania języka naturalnego. Prace powinny iść w kierunku m.in. rozszerzenia zbioru akceptowanych zdań języka angielskiego przez automat *PDAMS* o zdania złożone, opracowania formalnej reprezentacji zdań złożonych w bazie wiedzy, uwzględnienia kontekstu wypowiedzi, szczegółowego opracowania metody wnioskowania proceduralnego.

LITERATURA

1. Allen J. (1987) *Natural Language Processing*, California, Reading, Mass., The Benjamin/Cummings Publishing Company, Inc.
2. Arbib M.A. (1989) Schemas and Neural Networks for Sixth Generation Computing, *J. of Parallel Distributed Computing*, 6, 185-216.
3. Bandyopadhyay S. (1991) An Architecture for Distributed Knowledge Processing in Understanding Natural Language, *J. New Gener. Comput. Syst.*, 4(1), 59-84.
4. Bates M., Moser M.G., Stallard D. (1986) The IRUS transportable natural language database interface, w: L. Kerschberg (red.) *Expert Database Systems*, The Benjamin/Cummings Publishing Company, Inc., 617-630.
5. Bennett W.S., Slocum J. (1985) The LRC Machine Translation System (METAL), *Computational Linguistics*, 11(2-3), 111-121.
6. Berry-Rogghe G.L., Wulz H. (1978) The design of PLIDIS a problem solving information system with German as query language, w: L. Bolc (red.) *Natural Language Communication with Computers, Lecture Notes in Computer Science*, vol. 63, New York, Springer-Verlag.
7. Blaser A. (red.) (1988) *Natural Language at the Computer, Lecture Notes in Computer Science*, vol. 320, New York, Springer Verlag.
8. Blikle A. (1971) *Automaty i gramatyki. Wstęp do lingwistyki matematycznej*, Warszawa, PWN.
9. Bobrow D.G., Winograd T. (1977) An overview of KRL, a knowledge representation language, *Cognitive Science* 1(3), 3-46.
10. Bobrow R.J., Webber B.L. (1980) Knowledge representation for syntactic/semantic processing, *Proc. AAAI*, 316-323.
11. Brown J.S., Burton R.R. (1975) Multiple representations of knowledge for tutorial reasoning, w: D.G. Bobrow, A. Collins (red.) *Representation and Understanding*, New York, Academic Press.

12. Bünting K.D. (1985) *Wstęp do lingwistyki*, Warszawa, PWN.
13. Charniak E., Wilks Y. (1976) (red.) Computational Semantics, w: *Fundamental Studies in Computer Science*, vol. 4, Amsterdam, North-Holland Publishing Company.
14. Chomsky N. (1956) Three models for the description of language, *IRE Transactions PGIT*, 2, 113-124.
15. Chomsky N. (1962) *Context free grammars and pushdown storage*, Cambridge, MA, Quart. Prog. Rept. No 65, MIT Res. Lab. Electronics.
16. Chomsky N. (1965) *Aspects of the Theory of Syntax*, Cambridge, Mass., MIT Press.
17. Chomsky N. (1972) *Studies on Semantics in Generative Grammar*, The Hague, Mouton.
18. Chomsky N. (1981) *Lectures on Government and Binding*, Dordrecht, Foris.
19. Chmiel K., Stokłosa J. (1989) Algorytmy kompilacji krzemowej, *Studia z Automatyki*, t. 15, Warszawa-Poznań, PWN, 5-46.
20. Codd E.F. (1978) How About Recently?, w: B. Shneiderman (red.) *Databases: Improving usability and responsiveness*, New York, Academic Press.
21. Colmerauer A. (1978) Metamorphosis grammars, w: L. Bolc (red.) *Natural Language Communication with Computers*, New York, Springer-Verlag.
22. Conklin J. (1987) Hypertext: An introduction and survey, *IEEE Comput.*, 20(9), 17-41.
23. Dale R., Mellish Ch., Zock M. (1990) *Current Research in Natural Language Generation*, London, Academic Press.
24. DeJong G. (1982) An Overview of the FRUMP System, w: W.G. Lehnert, M.H. Ringle (red.) *Strategies for Natural Language Processing*, L. Erlbaum Associates, Hillsdale, N.J., 149-176.
25. Dillon G.L. (1977) *Introduction to Contemporary Linguistic Semantics*, New Jersey, Englewood Cliffs, Prentice-Hall, Inc.

26. Dobryjanowicz E. et al (1991) *System z dostępem w języku polskim. Stan prac*, 712, Warszawa, Prace IPI PAN.
27. Dowty D.R., Wall R.E., Peters S. (1981) *Introduction to Montague Semantics*, Dordrecht, Holland, D.Reidel.
28. Ducrot J.M. (1982) *The TITUS 4 System*, Institut Textile de France, Paris.
29. Durham J., Lamb D.A., Saxe S.B. (1983) Spelling Correction in User Interfaces, *Comm. ACM*, 26(10), 764-773.
30. Fahlman S.E. (1979) *NETL: a system for representing and using real world knowledge*, Cambridge, Mass., London, MIT Press.
31. Fillmore C. (1968) The case for case, w: E.Bach, R.T. Harms (red.) *Universals in Linguistic Theory*, New York, Holt, Reinchart and Winston, 1-90.
32. Foster J.M. (1976) *Przetwarzanie struktur listowych*, Warszawa, PWN.
33. Gazdar G., Klein E., Pullum G., Sag I. (1985) *Generalized Phrase Structure Grammar*, Cambridge, Mass., Harvard University Press.
34. Grosz B.J. (1983) TEAM, a Transportable Natural Language Interface System, w: *Proc. of the Conference on Applied Natural Language Processing*, Santa Monica, ACL & NRL, CA.
35. Guida G., Tasso C. (1983) IR-NLI: An Expert Natural Language Interface to Online Data Bases, w: *Proc. of the Conference on Applied Natural Language Processing*, Santa Monica, ACL & NRL, , CA.
36. Halliday M.A.K. (1967) Notes on transivity and theme in English, *J. of Linguistics*, 3, 4.
37. Hausser R. (1989) *Computation of language*, New York, Springer-Verlag.
38. Heidorn G.E. et al (1982) The EPISTALE Text-Critiquing System, *IBM Systems Journal*, 21(3), 305-326.

39. Hendrix G.G. et al (1978) Developing a Natural Language Interface to Complex Data, *ACM TODS*, 3(3), 105-147.
40. Hobbs J.R., Shieber S.M. (1987) An algorithm for generating quantifier scopings, *Computational Linguistics*, 13(1-2).
41. Hovy E.H. (1988) Generating Language with a Phrasal Lexicon, w: D.D. McDonald, L. Bolc (red.) *Natural Language Generation Systems*, New York, Springer -Verlag.
42. Hundt M.G. (1982) Working with the Weidner Aided Translation System, w: V. Lawson (red.) *Practical Experience of Machine Translation*, Amsterdam, North Holland Publishing Co.
43. INTELLECT Qery System User's Guide (1980) , AI Corp., Newton Center, MA.
44. Isabelle P., Bourbeau L. (1985) TAUM-Aviation: Its Techical Features and Some Experimental Results, *Computational Linguistics*, 11(1), 18-27.
45. Ishikawa H. et al (1987) Designing a knowledge-based natural language interface, *IEEE Expert*, 2(2), 57-71.
46. Jackendoff R.S. (1977) *X-Syntax: a study of phrase structure*, Cambridge, Mass., MIT Press.
47. Jensen K. (1988) Issus in Parsing, w: A. Blaser (red.) *Natural Language at the Computer*, *Lecture Notes in Computer Science*, vol. 320, New York, Springer Verlag.
48. Kanerva P. (1988) *Sparse Distributed Memory*, The MIT Press.
49. Kaplan J.S. (1984) Designing a Portable Natural Language Database Query System, *ACM TODS*, 9(1), 1-19.
50. Kay M. (1979) Functional grammar, w: *Proc. of the Fifth Annual Meating of the Berkeley Linguistic Society*.
51. Kazimierczak J. (1978) *System cybernetyczny*, Warszawa, Wyd. Wiedza Powszechna.

52. Kazimierczak J. (1979) *Elementy syntezy formalnej systemów operacyjnych*, Wrocław, Wydawnictwo Politechniki Wrocławskiej.
53. Kazimierczak J. (1985) Acquisition of Knowledge and Its Integration in Knowledge Base of the Rule-Based Expert System, w: *Paper on Science of Science and Forecasting*, No. 3-4, Wrocław, Wrocław Technical University Press, 23-38.
54. Kazimierczak J. (1987) Systemy ekspertowe z bazowaniem reguł - organizacja i zastosowanie, *Krajowa Konferencja Naukowo-Techniczna. Zastosowanie komputerów w przemyśle*, t. 2, Szczecin.
55. Kazimierczak J. (1989) Symbolic Representation and Understanding of Natural Language Sentences in the Knowledge-Based System, *Report No. 37/89*, Wrocław, ICT Wrocław Technical University Press.
56. Kazimierczak J. (1990) An Approach to Natural Language Processing in the Rule-Based Expert System, *Proc. of the ACM Computer Science'90 Conference*, Washington DC, New York, 215-222.
57. Kazimierczak J. (1991) Formal Representation of Natural Language Sentences in the Knowledge Base, *Report No. 41/91*, Wrocław, ICT Wrocław Technical University Press.
58. Kazimierczak J. (1992) Splitting Natural Language Sentences Into Groups of Words for Purposes of Automatic Programming, *Report No. 37/92*, Wrocław, ICT Wrocław Technical University Press.
59. Kazimierczak J. (1993) Transformation of Natural Language Sentences into Elements of Knowledge Base for Purpose of Automatic Programming, *Report No. 3/93*, Wrocław, ICT Wrocław Technical University Press.
60. Kolodner J.L. (1983) Indexing and Retrieval Strategies for Natural Language Fact Retrieval, *ACM TODS*, 8(3), 434-464.
61. Kowalski S., Mostowski A.W. (1979) *Teoria automatów i lingwistyka matematyczna*, Warszawa, PWN.
62. Kowalski R. (1989) *Logika w rozwiązywaniu zadań*, Warszawa, WNT.

63. Kulikowski J.L. (1986) *Zarys teorii grafów. Zastosowania w technice*, Warszawa, PWN.
64. Langacker R.W. (1987) *Foundations of Cognitive Grammar, vol. 1, Theoretical Prerequisites*, Stanford Univeristy Press.
65. *Longman Dictionary of Contemporary English* (1987), Longman Group UK Limited.
66. Luckhardt H.D. (1982) SUSY - Capabilities and Range of Application, *Multilingua*, 1-4, 213-219.
67. Lytinen S.L. (1986) Dynamically combining syntax and semantics in natural language processing, *Proc. AAAI*, 574-578.
68. McDonald D.D., Bolc L. (1988) *Natural Language Generation Systems*, New York, Springer-Verlag.
69. McKeown K.R. (1985) Discourse strategies for generating natural-language text, *AI*, 27(1), 1-41.
70. Miikkulainen D., Dyer M.G. (1991) Natural Language Processing with modular PDP networks and distributed lexicon, *Cognitive Science*, 15(3), 343-399.
71. Mikołajczak B. (red.) (1985) *Algebraiczna i strukturalna teoria automatów*, Warszawa-Lódź, PWN.
72. Mildred L., Shaw G., Woodward J. (1990) Modelling expert knowledge, *Knowledge Acquisition*, 179-206.
73. Montague R. (1970) Universal Grammar, *Theoria* 36.
74. Nebel B. (1990) Reasoning and Revision in Hybrid Representation Systems, w: J. Siekman (red.) *Lecture Notes in Artificial Intelligence*, New York, Springer-Verlag.
75. Nenov V. (1991) Perceptually Grounded Language Acquisition: A Neural/Procedural Hybrid Model, *Technical Report UCLA-AI-91-07*, University of California at Los Angeles.

76. Nielsen J. (1990) *Hypertext and Hypermedia*, San Diego, Calif., Academic Press.
77. Paice C.D. (1981) Automatic Generation of Literature Abstracts - An Approach Based on the Identification of Self Indicating Phrases, w: R.N. Oddy at al (red.) *Information Retrieval Research*, London, Butterworths, 172-191.
78. Pereira F.C.N., Warren D.H.D. (1980) Definite clause grammars for language analysis - A survey of the formalism and a comparison with augmented transition networks, *Artificial Inteligence* 13, 3, 231-278.
79. Ram A. (1992) Natural Language Understanding Information - Filtering Systems, *Comm. ACM*, 35(12), 80-81.
80. Revesz G.E. (1983) *Introduction to formal languages*, New York, McGraw-Hill Book Company.
81. Salton G. (1989) *Automatic Text Processing. The Transformation, Analysis, and Retrieval if Information by Computer*, Addison-Wesley Publishing Company, Reading, Mass.
82. Schank R.C., Colby K.M. (1973) *Computer Models of Thought and Language*, San Francisco, CA, Freeman.
83. Schank R.C. (1975) *Conceptual Information Processing*, Amsterdam, North Holland Publishing Co.
84. Schank R.C. (1982) Reminding and Memory Organization: An Introduction to MOPS, w: W.G. Lehnert, M.H. Ringle (red.) *Strategies for Natural Language Processing*, L. Erlbaum Associates, Hillsdale, N.J., 455-493.
85. Shapiro E., Warren D.H.D (1993) The Fifth Generation Project: Personal Perspectives, *Comm. ACM*, 36(3), 46-49.
86. Smólska J. (1976) *Gramatyka języka angielskiego*, Warszawa, Wydawnictwa Szkolne i Pedagogiczne.

87. Sowa J. (1991) Towards the expressive power of natural language, w: Sowa J. (red.) *Principles of Semantic Networks: Exploration in the Representation of Knowledge*, Morgan Kaufman Pub.
88. Tennant H. (1980) *Syntactic Analysis in JETS*, Working Paper - 26, Coord. Sc. Lab., University of Illinois, Urbana, Illinois 61801.
89. Thompson B.H., Thompson F.B. (1975) Practical language processing, the REL system as a prototype, w: *Advances in computers*, vol. 13, New York, Academic Press.
90. Thompson B.H., Thompson F.B. (1985) ASK is Transportable in Half a Dozen Ways, *ACM TOOLS*, 3(2), 185-203.
91. Thouin B. (1982) The METEO System, w: V. Lawson (red.) *Practical Experience of Machine Translation*, Amsterdam, North Holland Publishing Co.
92. Unold O. (1992a) Komputerowa analiza i synteza gramatyczna angielskiej grupy werbalnej osobowej, *Raport nr 2/92*, Wrocław, Wydawnictwo Politechniki Wrocławskiej.
93. Unold O. (1992b) For natural language processing, *Abstr. Systems Science XI'92*, Wrocław, Wrocław Technical University Press.
94. Unold O. (1992c) Model formalny automatycznej dekompozycji zdań języka naturalnego na elementy bazy wiedzy dla systemu z bazowaniem wiedzy, *II Ogólnopolska konferencja CIR'92*, Warszawa-Siedlce, 55-63.
95. Unold O. (1993) Formalna reprezentacja angielskiej grupy werbalnej w bazie wiedzy systemu z dostępem w języku naturalnym, *Materiały II Krajowej Konferencji Naukowej "Inżynieria Wiedzy i Systemy Ekspertowe"*, t. 2, Wrocław, 289-297.
96. Waltz D.L. (1978) An English Language Question Answering System for a Large Relational Database, *Comm. ACM*, 21(7), 526-539.
97. Warwick S. (1987) Automated Lexical Resources in Europe, *Lexicon Workshop LEX*, Stanford University.

98. Weicker R. (1974) Turing machines with associative memory access, w: J. Loeckx (red.) *Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 14*, New York, Springer-Verlag, 458-480.
99. Weizenbaum J. (1966) ELIZA, *Comm. ACM*, 9,36-45.
100. Winograd T. (1972) *Understanding Natural Language*, New York, Academic Press.
101. Winograd T. (1983) *Language as a Cognitive Process, vol. I, Syntax*, Reading, MA, Addison-Wesley.
102. Woods W.A. (1970) Transition network grammars for natural language analysis, *Comm. ACM*, 13, 591-606.
103. Woods W.A. (1977) Lunar rocks in natural English: Explorations in natural language question answering, w: A. Zampoli (red.) *Linguistics Structures Processing*, New York, Elsevier North-Holland.

SKOROWIDZ SYMBOLI I OZNACZEŃ

(e), 55	G' , 22
$(W+T+L+p)^*$, 19	G'^+ , 26
$(W+T+L)^*$, 19	$G(l_1)$, 25
$\rightarrow$, 18	$G(l_2)$, 25
$\oplus$, 30	$G(l_3)$, 25
α_0 , 18	$G(l_4)$, 25
α_i , 18	$G(l_5)$, 25
δ , 17	$G(l_6)$, 25
Φ , 18	G^+ , 25
Θ , 18	$G^+(k_i)$, 53
A, 18	$G^+(l_1)$, 27
A', 22	$G^+(l_2)$, 27
$A(l_1)$, 24	$G^+(l_3)$, 28
$A(l_2)$, 24	$G^+(l_4)$, 29
$A(l_3)$, 24	$G^+(l_5)$, 29
$A(l_4)$, 24	$G^+(l_6)$, 29
$A(l_5)$, 24	G^{++} , 30
$A(l_6)$, 24	GWO, 49
AI, 7	h, 19
B, 18	i_1 , 19
B', 22	i_2 , 19
C, 19	i_3 , 19
C', 22	i_4 , 20
ciąg (w), 13	i_5 , 20
podciąg (w) _i , 13	i_6 , 21
D, 19	i_7 , 21
D', 22	i_k , 23
d, 17	J, 19
d_m , 53	K, 19
d_p , 53	$K(q_a)$, 22
E, 19	K^* , 19
E', 22	$K^+(k_1)$, 56
$E^+(e_1)$, 58	$K^+(k_2)$, 56
E^+_{ext} , 58	$K^+(k_3)$, 56
F, 17	$K^+(k_4)$, 56
G, 19	$K^+(k_5)$, 56

$K^+(k_6)$, 56	l_2 , 16
$K^+(k_7)$, 56	l_t , 20
$K^+(k_8)$, 56	M , 18
$K^+(k_i)$, 57	M_i , 17
K^+_{ext} , 57	m_h , 17
K_0 , 18	m_l , 17
K_1 , 19	m_r , 17
$K_1 \Rightarrow K_2$, 19	N , 18
K_2 , 19	N' , 22
K_a , 20	NLP , 1
$K^a \Rightarrow K_{nas}$, 22	O , 18
KB , 11	P , 18
K_f , 18	P' , 22
K^a , 21	<i>PDAMS</i> , 15
$K_i \Rightarrow K_{i+1} \Rightarrow K_{i+2}$, 23	p , 17
K_i , 23	p -marker, 16
K_{i+1} , 23	Q , 17
K_{i+2} , 23	Q^* , 18
K_{nas} , 22	q_0 , 17
K_t , 18	q_a , 18
$K_t \Rightarrow^* K_{t+1}$, 18	q_f , 17
$K_t \Rightarrow K_{t+j}$, 18	q_i , 19
K_{t+1} , 18	q_k , 21
k_i' , 55	q_x , 22
k_1' , 55	R , 19
k_2' , 55	r_0 , 27
k_3' , 55	r_2 , 27
k_4' , 55	r_9 , 27
k_5' , 55	r_1 , 25
k_6' , 55	r_3 , 25
k_7' , 55	r_4 , 25
k_8' , 55	r_5 , 25
kss , 36	r_6 , 27
k_j , 51	r_7 , 27
L , 16	r_8 , 29
L^* , 19	r_{p^i5qklp} , 23
l -marker, 16	$r_{p^i k}$, 23
l_1 , 16	r_p , 23

S , 19	w_{jh+1} , 20
S_1 , 16	w_{jk} , 19
S_2 , 16	w_{jk+1} , 19
S_3 , 16	$w_{t^i s q k l p}$, 23
S_4 , 16	$w_{t^i k}$, 23
S_5 , 16	w_t , 23
S_6 , 16	X , 19
s_1 , 72	x , 18
s_2 , 72	Y , 20
s_3 , 72	y , 18
s_4 , 72	Z , 17
s_5 , 72	Z_1 , 17
słownik D' , 53	Z_1^* , 19
słownik D'' , 54	Z_2 , 17
słownik E , 58	Z_2^* , 19
słownik K , 57	Z_3 , 17
T , 16	Z_4 , 17
t , 18	Z_5 , 17
t-marker, 16	Z_6 , 17
t_1 , 16	Z_6^* , 19
t_2 , 16	z_0 , 17
U , 18	z_{00} , 17
v , 18	z_{01} , 17
W , 17	z_{02} , 17
W^* , 18	z_{03} , 16; 17
w_0 , 12	z_{04} , 17
w_1 , 12	z_{05} , 17
w_2 , 12	z_{06} , 16
w_3 , 12	z_1 , 58
w_4 , 13	z_3 , 53
w_5 , 13	z_4 , 57
w_6 , 13	z_5 , 57
w_7 , 13	z_6 , 57
w_8 , 13	z_7 , 57
w_9 , 13	z_9 , 58
w_{jh} , 20	z_k , 53

ZALĄCZNIK A

Tabulogram programu AUTOMAT stanowiącego realizację programową automatu *PDAMS* (por. rozdział 2.4).

////////////////////////////////////PLIK AUTOMAT.H

```
#define Uses_TKeys
#define Uses_TApplication
#define Uses_TProgram
#define Uses_TEvent
#define Uses_TRect
#define Uses_TDialog
#define Uses_TStaticText
#define Uses_TButton
#define Uses_TMenuBar
#define Uses_TSubMenu
#define Uses_TMenuItem
#define Uses_TStatusLine
#define Uses_TStatusItem
#define Uses_TStatusDef
#define Uses_TDeskTop

#define Uses_MsgBox
#define Uses_TInputLine
#define Uses_TSortedListBox
#define Uses_TListViewer
#define Uses_TStringCollection
#define Uses_TCollection
#define Uses_TNSCollection
#define Uses_TView
#define Uses_TScreen
#define Uses_TProgram
#define Uses_TPoint
#define Uses_TBackground
#define Uses_opstream
#define Uses_ipstream
#define Uses_TScroller
#define Uses_TScrollBar
#define Uses_TRadioButton
#define Uses_TSItem
#define Uses_TCluster
#define Uses_TLabel
#define Uses_TEventQueue

#include <tv.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>

const int cmEdycja      =100;
const int CzołowkaCmd   =101;
const int cmDosShell    =102;
const int cmVideoMode   =103;
const int cmPodzial     =104;
const int cmOkno        =105;
const int cmPiszwOkno   =106;
```

```
const int cmPOpcje      =107;
const int cmPWyniki     =108;
const int cmCloseAll    =109;
```

```
// Prototypy funkcji
```

```
int  pdams(void);
int  typ(char);
char * indeks(char *, char *);
char * symbol(char *, char *);
char * dajW(char *,char *);
char * dajI(char *,char *);
char * dajQ(char *,char *);
char * dajluk(char *,char *);
char * pomin(char *);
char * szukaj(char *, char *);
char * czytajTW(char *);
void przesunTW(void);
char * przejście(char *,char *);
char * alterluk(char *, char *);
char * znajdz_luk(char *,char *);
void push(char *(*)[],int*, char *);
char * pop(char *(*)[],int*,char *);
char * zestosu(char* (*)[],int);
char * i1(char*);
char * i2(char*);
char * i3(char*);
char * i4(char*);
char * i5(char*);
char * i6(char*);
char * i7(char*);
void wypelnf(void);
int  czylt(char *);
int  czyf(char *);
char * dajL(char *,char *);
void stanpocz();
void blad(int);
void druch(char *, char *, char *);
```

```
void dakcja(char *);
void dalter(char *);
void podzialW(void);
void tworzą_kss(void);
void komponent(int);
```

```
////////////////////////////////////KONIEC PLIKU AUTOMAT.H
```

```
////////////////////////////////////PLIK AUTOMAT.CPP
```

```
#include<stdio.h>
#include<ctype.h>
#include<alloc.h>
#include<string.h>
#include<process.h>
#include<stdlib.h>
#include<conio.h>
#include<dos.h>
#include "automat.h"
```

```
const int maxLines=100    ;
```

```

extern char *lines[];
extern char *plines[];
extern char *tlines[];
extern char *alines[];
extern char *aplines[];
extern char *wlines[];
extern char *slowa[];
extern int wlineCount;
extern int aplineCount;
extern int sCount;
extern class TOknoH: public TWindow{} ;
extern TOknoH *OknoPomoc;
extern int lineCount;
extern char ciagW[];
extern ushort trybPracy;
extern void komunikat(char*);
extern pomoc();

```

```

enum typy{          // Typy znakow w wyrazeniu symbolicznym
    l_nawias,
    p_nawias,
    przecinek,
    cyfra,
    litera
};

```

```

char* (*tabfun[8])(char *); // Tablica funkcji

```

```

#define LNAWIAS  '('
#define PNAWIAS  ')'
#define PRZECINEK  ','
#define STR_KONIEC  '\0'
#define L1      "11"
#define L2      "12"
#define L3      "13"
#define L4      "14"
#define L5      "15"
#define L6      "16"
#define P      "p"

```

```

#define LMARKER  "("
#define PMARKER  ")"

```

```

char *l_1=          // G(11) - grupa nominalna
"q21(w3i1q21,w7i1q22(w2i1q23,w6i1q27(w1i7q28,w2i7q28,w3i7q28,\
w4i7q28,w5i7q28,w6i7q28,w7i7q28,w8i7q28,w0i7q28),w2i7q28,\
w3i7q28,w4i7q28,w5i7q28,w6i7q28,w7i7q28,w8i7q28,w0i7q28)),\
w1i1q23,w2i1q23,w3i5q24q61,w4i5q24q61,w6i5q24q61),q24(w1i2q25(w1i1q23,\
w3i5q26q61,w4i5q26q61,w6i5q26q61),w2i2q25,w3i2q25,w4i2q25,w5i2q25,w6i2q25,\
w7i2q25,w8i2q25,w0i2q25),q26(w1i2q25,w2i2q25,w3i2q25,w4i2q25,w5i2q25,w6i2q25,\
w7i2q25,w8i2q25,w0i2q25)";

```

```

char *l_2=          // G(12) - grupa werbalna
"q41(w5i1q42(w5i1q44(w5i1q44,w1i7q45,w2i7q45,w3i7q45,w4i7q45,\
w6i7q45,w7i7q45,w8i7q45,w0i7q45),w6i1q43(w5i1q44),w1i7q45,\
w2i7q45,w3i7q45,w4i7q45,w6i7q45,w7i7q45,w8i7q45,w0i7q45))";

```

```

char *l_3=          // G(13) - grupa przymiotnikowa
"q61(w3i5q62q81,w6i5q62q81,w4i1q63(w1i7q66,w2i7q66,w3i7q66,w4i7q66,\
w5i7q66,w6i7q66,w7i7q66,w8i7q66,w0i7q66,w6i1q65(w1i7q66,\

```

```
w2i7q66,w3i7q66,w4i7q66,w5i7q66,w6i7q66,w7i7q66,w8i7q66,\
w0i7q66)),q62(w1i2q64(w4i1q63,w5i1q63),w2i2q64,w3i2q64,w4i2q64,w5i2q64,\
w6i2q64,w7i2q64,w8i2q64,w0i2q64)";
```

```
char *_4= // G(14) - grupa przyslowkowa
"q81(w3i1q82(w1i1q83(w6i1q83,w1i7q84,w2i7q84,w3i7q84,w4i7q84,\
w5i7q84,w7i7q84,w8i7q84,w0i7q84)),w6i1q83)";
```

```
char *_5= // G(15) - grupa przyimkowa
"q101(w7i1q102(w6i1q105(w1i7q106,w2i7q106,w3i7q106,w4i7q106,\
w5i7q106,w6i7q106,w7i7q106,w8i7q106,w0i7q106),w1i5q104q21,\
w2i5q104q21,w3i5q104q21,w4i5q104q21,w6i5q104q21,w7i5q104q21),\
w6i1q103(w7i1q102)),q104(w1i2q105,w2i2q105,w3i2q105,w4i2q105,w5i2q105,\
w6i2q105,w7i2q105,w8i2q105,w0i2q105)";
```

```
char *_6= // G(16) - grupa werbalna nieosobowa
"q121(w7i1q122(w5i1q124(w5i1q124,w1i7q125,w2i7q125,w3i7q125,w4i7q125,\
w6i7q125,w7i7q125,w8i7q125,w0i7q125)),w6i1q123(w7i1q122))";
```

```
char *_0= // G(10) - zdanie proste
"q0(w1i5q2q21,w2i5q2q21,w3i5q2q21,w4i5q2q21,w6i5q2q21,w7i5q2q21,w6i5q0q121,\
w7i5q0q121),q2(w5i5q3q41,w6i5q4q101,w7i5q4q101),q3(w0i1q10,w6i5q3q101,\
w7i5q3q101,w1i5q3q21,w2i5q3q21,w3i5q3q21,w4i5q3q21,w6i5q3q21,w7i5q3q21,\
w3i5q3q61,w4i5q3q61,w6i5q3q61,w3i5q3q81,w6i5q3q81,w6i5q3q121,w7i5q3q121),\
q4(w5i5q3q41,w6i5q4q101,w7i5q4q101)";
```

```
static char *wyr_symbol; // Wyrażenie Symboliczne opisujące strukturę
// automatu
```

```
const int maxStos = 100;
```

```
char *stos1[maxStos], //Stosy: S1
*stos2[maxStos], // % S2
*stos3[maxStos], // % S3
*stos4[maxStos], // % S4
*stos5[maxStos], // % S5
*stos6[maxStos], // % S6
*alterlisty[maxStos], // Stos list alternatywnych przejść automatu
parametr[10], // Parametr dla funkcji i5
*aktu_stan; // Aktualny stan automatu
```

```
int wierzchS1 = 0, // Wierzchołki stosów: S1
wierzchS2 = 0, // % S2
wierzchS3 = 0, // % S3
wierzchS4 = 0, // % S4
wierzchS5 = 0, // % S5
wierzchS6 = 0, // % S6
wierzch_al = 0, // Wierzchołek stosu 'alterlisty'
glowicaTW = 0; // Pozycja głowicy taśmy wejściowej
```

```
int krok = 1; // Numer kolejnego kroku automatu
```

```
const int maxKss = 50;
```

```
struct k { // tablica komponentów
char l[3];
char *komponenty;
char *slovo;
} kss[maxKss];
```

```
int wierzch ;
```

```

int licz_kss ;
int max_licz_kss;
int kolejka ;
char b[4]="k ";

// Programowa realizacja automatu PDAMS
int pdams(void)
{
char stanp[3] ="q0",           // stan początkowy automatu
  stank[4] ="q10",             // stan końcowy
  stan[5],                     // stan aktualny
  *luki =(char*)malloc(100),
  luk[20] ="",
  *wskluki =luki;

int i;

if(luki==NULL) blad(1);
wypelntf();
sprintf(wyr_symbol=(char*)malloc(strlen(l_0)+strlen(l_1)+strlen(l_2)+
strlen(l_3)+strlen(l_4)+strlen(l_5)+strlen(l_6)+20),
"%s,%s,%s,%s,%s,%s,%s,%s",l_0,l_1,l_2,l_3,l_4,l_5,l_6);
if(wyr_symbol==NULL) blad(1);
aktu_stan=stan;
strcpy(stan,stanp);

lines[lineCount++]= newStr("Krok   Ze stanu   Przejscie   Do stanu");
message(TProgram::deskTop, evBroadcast, cmPiszwOkno,0);

stanpocz();
strcpy(stan,stanp);
if(!strstr(ciaW,"w0"))
strcat(ciaW,"w0");
while(strcmp(stan,stank))//////////////////////////////////// krok 1
{
    luki=wskluki;
    if(czyf(stan))//////////////////////////////////// krok 2
    {
        dakeja("i6");
        i6(stan);
    }
    else
        if( strlen(alterluki(stan,luki)) )//////////////////////////////////// krok 3 i krok 4
        {
            if( strlen(luki+=strlen(dajluk(luki,luk))) )///////// krok 4a
            {

strcpy(alterlisty[++wierzch_al]=(char*)malloc(strlen(luki)+1),luki);
                if(alterlisty[wierzch_al]==NULL) blad(1);
                dakeja("i3");
                i3(stan);
            }
            druch(stan,luk,luki);
        }
    else
    {
        if(!wierzchS1)//////////////////////////////////// krok 5
        {
            blad(0);
            return -1;
        }
    }
}

```

```

        else
        {
            dakcja("i4");
            i4(stan);//////////////////////// krok 6
            if(
strlen(alterlisty[wierzch_al]+strlen(dajluk(alterlisty[wierzch_al],luk))) )
            {
                luki=alterlisty[wierzch_al];
                dakcja("i3");
                i3(stan);
            }
            else
            {
                luki=alterlisty[wierzch_al];
                free(alterlisty[wierzch_al]);
                alterlisty[wierzch_al--]=STR_KONIEC;
            }
            druch(stan,luk,alterlisty[wierzch_al]);
        }
    }

};
lines[lineCount++]= newStr(" ---> Automat osiagnal stan koncowy q10 <---");
message(TProgram::deskTop, evBroadcast, cmPiszwOkno,0);
return 0;
}

//f. zwraca typ znaku
int typ(char znak)
{
    switch(znak)
    {
        case LNAWIAS :      return l_nawias;
        case PNAWIAS :      return p_nawias;
        case PRZECINEK:      return przecinek;
        default :           return isdigit(znak) ? cyfra : litera;
    }
}

// f. z poczatku lancucha 'str' kopiuje indeks symbolu (liczbe) do 'id'
char *indeks(char *str,char *id)
{
    char *wsk =id;

    for(;isdigit(*str);*wsk++=*str++);
    *wsk=STR_KONIEC;
    return id;
}

// f. z poczatku lancucha 'str' kopiuje symbol (litera z jej indeksem)
// do 'term'-u
char *symbol(char *str, char *term)
{
    char *wsk =term;

    if( typ(str[0]) != litera )
        strcpy(term,"");
    else
    {
        *wsk++=str[0];
    }
}

```

```

        indeks(str+1,wsk);
    }
    return term;
}
// f. pobiera z termu str='wi ip qk' symbol wi
char *dajW(char *str,char *symb)
{
    return symbol(str,symb);
}

// f. pobiera z termu str='wi ip qk' symbol ip,
// w przypadku ip=i5 dodatkowo symbol qi (str='wi ip qi qk')
char *dajI(char *str,char *symb)
{
    char *wsk =str,
        buf[10];

    wsk+=strlen(dajW(wsk,buf));
    symbol(wsk,symb);
    if(!strcmp(symb,"i5"))
        strcat(symb,strcpy(parametr,symbol(wsk+strlen(symb),buf)));
    return symb;
}

// f. pobiera z termu str='wi ip qk' symbol qk
char *dajQ(char *str,char *symb)
{
    char *wsk =str,
        buf[20];

    return symbol(wsk+=(strlen(dajI(wsk,buf))+strlen(dajW(wsk,buf))),symb);
}

// f. pobiera z poczatku lancucha str luk 'wi ip qk'
char *dajluk(char *str,char *symb)
{
    char *wsk=str,
        symbW[10],
        symbI[20],
        symbQ[10];

    if(!str)
        return NULL;
    else
        return strcat(strcat(strcpy(symb,dajW(wsk,symbW)),dajI(wsk,symbI)),
            dajQ(wsk,symbQ));
}

// f. pomija zawartosc nawiasu (...)
char *pomin(char *str)
{
    char *wsk =str;
    int licz_naw=0;

    do
    {
        switch(typ(*wsk++))
        {
            case l_nawias:

```



```

        licz_naw++;
        break;
    case p_nawias:
        --licz_naw;
        break;
    }
} while(licz_naw);
return wsk;
}

// f. szuka pierwszego wystąpienia podlancucha 's' w lancuchu 'str'
char *szukaj(char *str, char *s)
{
    return strstr(str,s);
}

//f. czyta z TW-'ciagW' symbol 'wi' i przesuwá glowice TW-'glowicaTW'
// o dlugosc symbolu
char *czytajTW(char *symbolW)
{
    return symbol(ciagW+glowicaTW,symbolW);
}

// f. przesuwá 'ciagW' o dl. aktualnie czytanego symbolu
void przesunTW(void)
{
    char symbW[10];

    glowicaTW+=strlen(czytajTW(symbW));
}

// f. powoduje 'przejscie' po luku 'luk' = wi ip qk, zwraca kolejny stan
// automatu który jest stanem qk lub stanem ustawionym przez wywolana
// instrukcje ip
char *przejscie(char *luk, char *stan)
{
    char symbI[20],
        ind[5];

    dakeja(dajI(luk,symbI));
    return (*tabfun[atoi(indeks(dajI(luk,symbI)+1,ind))])(dajQ(luk,stan));
}

// f. tworzy liste 'luki' alternatywnych przejsc ze stanu 'stan'
// dla aktualnie czytanego symbolu wejsciowego
char *alterluki(char *stan, char *luki)
{
    char s_stan[20],
        lluki[100],
        *s_miejsce,
        luk[20],
        symbW[10];

    strcpy(lluki,"");
    s_miejsce=szukaj(wyr_symbol,strcat(strcpy(&s_stan[0],stan), "("));
    s_miejsce+=strlen(s_stan);

    s_miejsce=znajdz_luk(s_miejsce,czytajTW(symbW));
    if(!s_miejsce)
        strcpy(luki,"");
}

```

```

else
do
{
    strcat(lluki,dajluk(s_miejsce,luk));
    s_miejsce+=strlen(luk);
    s_miejsce=znajdz_luk(s_miejsce,czytajTW(symbW));
} while(s_miejsce);
return strcpy(luki,lluki);
}

```

```

char *znajdz_luk(char *str,char *symbW)
{
char *wsk =str,
    buf[10];

do
{
    switch(typ(*wsk))
    {
        case litera:
            if(!strcmp(symbol(wsk,buf),symbW))
                return wsk;
            else break;
        case p_nawias:
            return wsk=STR_KONIEC;
        case l_nawias:
            wsk=pomin(wsk)-1;
    }
    wsk++;
} while(1);
}

```

```

void push(char* (*s)[],int *wierzch,char *symb)
{
    (*s)[++*wierzch]=(char*)malloc(strlen(symb)+1);
    if((*s)[*wierzch]==NULL) blad(1);
    strcpy((*s)[*wierzch],symb);
}

```

```

char *pop(char* (*s)[],int *wierzch,char *symb)
{
    strcpy(symb,(*s)[*wierzch]);
    free((*s)[*wierzch]);
    (*s)[(*wierzch)--]=STR_KONIEC;
    return symb;
}

```

```

char *zestosu(char* (*s)[],int pozycja)
{
    return (*s)[pozycja];
}

```

```

char *il(char *stan)
{
char symbW[10];

    push(&stos3,&wierzchS3,czytajTW(symbW));
    przesunTW();
    return stan;
}

```

```

char *i2(char *stan)
{
char *symb =(char*)malloc(10),
    *stospom[100];
int wierzpom =0,
    j;

    if(symb==NULL) blad(1);
    for(j=0;j<100;j++)      stospom[j]=STR_KONIEC;
    while( !czylyt(pop(&stos6,&wierzchS6,symb)) )
        push(&stospom,&wierzpom,symb);
    push(&stospom,&wierzpom,symb);

    while( czylyt(pop(&stos6,&wierzchS6,symb)) )
        push(&stospom,&wierzpom,symb);
    push(&stos6,&wierzchS6,symb);

    while(wierzpom)
        push(&stos3,&wierzchS3,pop(&stospom,&wierzpom,symb));
    return stan;
}

char *i3(char *stan)
{
char *buf=(char *)malloc(10);
int i;

    if(buf==NULL) blad(1);
    push(&stos1,&wierzchS1,aktu_stan);
    push(&stos2,&wierzchS2,itoa(glowicaTW,buf,10));
    push(&stos6,&wierzchS6,P);
    push(&stos5,&wierzchS5,P);
    for(i=1;i<=wierzchS3;i++)
        push(&stos5,&wierzchS5,zestosu(&stos3,i));
    return stan;
}

char *i4(char *stan)
{
char *buf=(char*)malloc(20),
    *stospom[100];
int wierzpom =0,
    j,
    i    =0;

    if(buf==NULL) blad(1);
    aktu_stan=pop(&stos1,&wierzchS1,stan);
    glowicaTW=atoi(pop(&stos2,&wierzchS2,buf));
    while(wierzchS3) // zerowanie stosu S3
        pop(&stos3,&wierzchS3,buf); //

    while( strcmp(pop(&stos5,&wierzchS5,buf),P) ) // przepisanie z S5 na S3
    { //
        push(&stospom,&wierzpom,buf); //
        i++; //
    } //
    for(j=0;j<i;j++) //
        push(&stos3,&wierzchS3,pop(&stospom,&wierzpom,buf));

    while( strcmp(pop(&stos6,&wierzchS6,buf),P) );

```

```

        return stan;
    }

char *i5(char *stan)
{
    char symbL[10];

    push(&stos4,&wierzchS4,parametr);
    // t-marker
    push(&stos3,&wierzchS3,dajL(stan,symbL));
    push(&stos3,&wierzchS3,LMARKER);
    //
    return stan;
}

char *i6(char *stan)
{
    return pop(&stos4,&wierzchS4,stan);
}

char *i7(char *stan)
{
    char *buf=(char*)malloc(20),
        *stospom[100];
    int wierzpom =0,
        j;

    if(buf==NULL) blad(1);
    for(j=0;j<100;j++)    stospom[j]=STR_KONIEC;
    while(wierzchS3)
        push(&stospom,&wierzpom,pop(&stos3,&wierzchS3,buf));
    while(wierzpom)
        push(&stos6,&wierzchS6,pop(&stospom,&wierzpom,buf));
    // t-marker
    push(&stos6,&wierzchS6,PMARKER);
    //
    return stan;
}

// f. inicjuje tablice funkcji 'tabfun'
void wypeIntf(void)
{
    tabfun[1]=i1;
    tabfun[2]=i2;
    tabfun[3]=i3;
    tabfun[4]=i4;
    tabfun[5]=i5;
    tabfun[6]=i6;
    tabfun[7]=i7;
}

// f. sprawdza czy 'symb' nalezy do zbioru L={l1,l2,l3,l4,l5,l6}
int czyIt(char *symb)
{
    if(!strcmp(symb,L1) || !strcmp(symb,L2) || !strcmp(symb,L3) || !strcmp(symb,L4) ||
        !strcmp(symb,L5) || !strcmp(symb,L6))
        return 1;
    else
        return 0;
}

```

```
// f. sprawdza czy 'symb' nalezy do zbioru F={q28,q45,q66,q84,q106,q125}
int czyf(char *symb)
```

```
{
    if(!strcmp(symb,"q28") || !strcmp(symb,"q45") || !strcmp(symb,"q66") || !strcmp(symb,"q84")
        || !strcmp(symb,"q106") || !strcmp(symb,"q125"))
        return 1;
    else
        return 0;
}
```

```
// f. zwraca przyporządkowany stanowi początkowemu podautomatu symbol li
char* dajL(char *symb,char *symbL)
```

```
{
    if(!strcmp(symb,"q21"))
        return strcpy(symbL,L1);

    if(!strcmp(symb,"q41"))
        return strcpy(symbL,L2);

    if(!strcmp(symb,"q61"))
        return strcpy(symbL,L3);

    if(!strcmp(symb,"q81"))
        return strcpy(symbL,L4);

    if(!strcmp(symb,"q101"))
        return strcpy(symbL,L5);

    if(!strcmp(symb,"q121"))
        return strcpy(symbL,L6);

    return NULL;
}
```

```
// f. ustawia automat w jego stan początkowy
void stanpocz(void)
```

```
{
    int j;

    for(j=0;j<100;j++)
    {
        free(stos1[j]);   stos1[j]=STR_KONIEC;
        free(stos2[j]);   stos2[j]=STR_KONIEC;
        free(stos3[j]);   stos3[j]=STR_KONIEC;
        free(stos4[j]);   stos4[j]=STR_KONIEC;
        free(stos5[j]);   stos5[j]=STR_KONIEC;
        free(stos6[j]);   stos6[j]=STR_KONIEC;
        free(alterlisty[j]);alterlisty[j]=STR_KONIEC;
    }
    wierzchS1 = 0;
    wierzchS2 = 0;
    wierzchS3 = 0;
    wierzchS4 = 0;
    wierzchS5 = 0;
    wierzchS6 = 0;
    wierzch_al = 0;
    głowicaTW = 0;
    krok      = 0;
}
```

```

void blad(int numer)
{
    switch(numer)
    {
        case 0:
            komunikat("nie ma przejścia - błąd syntaktyczny!");
            break;
        case 1:
            komunikat("za mało pamięci operacyjnej!");
            exit(1);
    };
}

void druch(char *stan, char *luk, char *luki)
{
    int i;
    char sstan[10]="";
    char buf[140];

    strcpy(sstan,stan);
    buf[0]=0;
    sprintf(buf,"%s",ciagW);
    tlines[0] = newStr(buf);

    memset(&buf[0],'-',80);
    strnset(&buf[0],' ',glowicaTW);
    buf[glowicaTW+1]=0;
    tlines[1] = newStr(buf);

    buf[0]=0;
    sprintf(buf,"%s","Stos 1: ");
    for(i=0;i<wierzchS1;i++) sprintf(&buf[strlen(buf)],"%s ",zestosu(&stos1,i+1));
    plines[0] = newStr(buf);

    buf[0]=0;
    sprintf(buf,"%s","Stos 2: ");
    for(i=0;i<wierzchS2;i++) sprintf(&buf[strlen(buf)],"%s ",zestosu(&stos2,i+1));
    plines[1] = newStr(buf);
    buf[0]=0;

    sprintf(buf,"%s","Stos 3: ");
    for(i=0;i<wierzchS3;i++) sprintf(&buf[strlen(buf)],"%s ",zestosu(&stos3,i+1));
    plines[2] = newStr(buf);
    buf[0]=0;

    sprintf(buf,"%s","Stos 4: ");
    for(i=0;i<wierzchS4;i++) sprintf(&buf[strlen(buf)],"%s ",zestosu(&stos4,i+1));
    plines[3] = newStr(buf);
    buf[0]=0;

    sprintf(buf,"%s","Stos 5: ");
    for(i=0;i<wierzchS5;i++) sprintf(&buf[strlen(buf)],"%s ",zestosu(&stos5,i+1));
    plines[4] = newStr(buf);
    buf[0]=0;

    sprintf(buf,"%s","Stos 6: ");
    for(i=0;i<wierzchS6;i++) sprintf(&buf[strlen(buf)],"%s ",zestosu(&stos6,i+1));
    plines[5] = newStr(buf);

    buf[0]=0;

```

```

    sprintf(buf, " %3d)%10s %15s", krok++, sstan, luk);

    dalter(luki);
    przejście(luk, stan);
    if(czyf(stan))
        sprintf(&buf[strlen(buf)], "%10s", stan);
    else
        sprintf(&buf[strlen(buf)], "%10s", stan);
    lines[lineCount++] = newStr(buf);
    message(TProgram::deskTop, evBroadcast, cmPiszwOkno, 0);
}

void dakcja(char *akcja)
{
    char buf[80] = "";

    sprintf(buf, "%s", "Instrukcja: ");
    if(strlen(akcja) > 2)
    {
        strncat(buf, akcja, 2);
        alines[0] = newStr(buf);
        buf[0] = 0;
        sprintf(buf, "%s", "Argument : ");
        strcat(buf, &akcja[2]);
        alines[1] = newStr(buf);
    }
    else
    {
        strcat(buf, akcja);
        alines[0] = newStr(buf);
        alines[1] = 0;
    }
    if(!trybPracy)
        getch();
}

void dalter(char *lista)
{
    char *luki = lista;
    char luk[20], buf[80];

    for( int i = 0; i < maxLines; i++ ) { delete aplines[i]; aplines[i] = NULL; } aplineCount = 0;
    aplines[aplineCount++] = newStr("Alter. przejścia.");

    while( strlen(dajluk(luki, luk)) )
    {
        sprintf(buf, "%s", luk);
        aplines[aplineCount++] = newStr(buf);
        luki += strlen(luk);
    }
};

void podzialW(void)
{
    char buff[140];
    char buf[5];
    int i = 0;
    char *poz;

    for(i = 0; i < maxKss; i++)

```

```

    {
        delete kss[i].komponenty;
        delete kss[i].slowo;
        kss[i].komponenty=NULL;
        kss[i].slowo=NULL;
        kss[i].l[0]=STR_KONIEC;
    }

    wierzch = 1;
    licz_kss = 1;
    max_licz_kss = 0;
    kolejka = 0;

    kss[0].komponenty=(char*)malloc(50);
    kss[0].komponenty[0]=STR_KONIEC;
    do{
        buf[0]=STR_KONIEC;
        strcpy(buf,zestosu(&stos6,wierzch++));
        switch(buf[0])
        {
            case 'l':
                licz_kss=++max_licz_kss;
                itoa(licz_kss,&b[1],10);
                if(strlen(&b[1]) == 1) // indeks przy k/w jest 2-cyfrowy
                    b[2]=' ';
                strcat(kss[0].komponenty,b);

                kss[licz_kss].komponenty=(char*)malloc(50);
                kss[licz_kss].komponenty[0]=STR_KONIEC;
                strcpy(kss[licz_kss].l,buf);
                tworz_kss();
                break;

            default:
                break;
        };
    } while(wierzch <= wierzchS6);

    // przypisanie slow do komponentow
    komponent(0);

    // wydruk
    wlines[wlineCount++]=newStr("PODZIAL CIAGU (w) NA KOMPONENTY
    STRUKTURALNO-SEMANTYCZNE:");
    sprintf(buff," (w) <-> %s",kss[0].komponenty);
    wlines[wlineCount++]=newStr(buff);
    wlines[wlineCount++]=newStr("");
    wlines[wlineCount++]=newStr("Definicja komponentow strukturalno-semantycznych:");

    i=1;
    do{
        sprintf(buff," k%-2d(%s) <-> %s (%s)",i,kss[i].l,kss[i].komponenty,kss[i].slowo);
        wlines[wlineCount++]=newStr(buff);
    } while(++i <= max_licz_kss);

}

void tworz_kss(void)
{
    int licz_naw=0;

```



```

int temp;
char buf[5];
do{
    buf[0]=STR_KONIEC;
    strcpy(buf,zestosu(&stos6,wierzch++));
    switch(buf[0])
    {
        case 'l':
            if (licz_naw == 1)
            {
                temp=licz_kss;
                licz_kss=++max_licz_kss;
                itoa(licz_kss,&b[1],10);
                if(strlen(&b[1]) == 1)
                    b[2]=' ';
                strcat(kss[temp].komponenty,b);
                kss[licz_kss].komponenty=(char*)malloc(50);
                kss[licz_kss].komponenty[0]=STR_KONIEC;
                strcpy(kss[licz_kss].l,buf);
                tworz_kss();
                licz_kss=temp;
            }
            break;
        case '(':
            ++licz_naw;
            break;
        case ')':
            --licz_naw;
            break;
        case 'w':
            if (licz_naw == 1)
                strcat(kss[licz_kss].komponenty,strcat(buf," "));
            break;
        case 'p':
            break;
    };
}while(licz_naw);
}

```

```
void komponent(int i)
{
    char *poz;
    char buf[4];

    poz=kss[i].komponenty;
    kss[i].slowo=(char*)malloc(100);
    kss[i].slowo[0]=STR_KONIEC;
    do{
        if(*poz == 'w')
            strcat(strcat(kss[i].slowo," "),newStr(slowa[kolejka++]));
        else
        {
            strcat(strcat(kss[i].slowo," "),newStr("*"));
            komponent(atoi((char*)memcpy(strcpy(buf," "),poz+1,2)));
        }
        poz+=3;
    } while(*poz);
    return;
}
```

////////////////////////////////KONIEC PLIKU AUTOMAT.CPP

```

//////////////////////////////////////PLIK MENU.CPP
#include "automat.h"

```

```
extern krok ;
```

```

// kategoria syntakt. slowa
ushort katslowa;           // 0 - w1, ..., 7 - w8
ushort trybPracy;
// zdanie wejscowe
char zdanie[120];

```

```

// Tasma wejscowa
char ciagW[120]="";

```

```

const int maxLineLength = maxViewWidth+1;
const int maxLines      = 100;
char *lines[maxLines];
int lineCount = 0;

```

```

char *wlines[maxLines];
int wlineCount = 0;

```

```

char *aplines[maxLines];
int aplineCount = 0;

```

```

char *slowa[maxLines];
int sCount=0;

```

```

char *plines[10];
int plineCount = 6;

```

```

char *hlines[maxLines];
int hlineCount = 0;

```

```

char *tlines[2];
char *alines[2];

```

```

int lkat(char*);
char *symbolW(ushort);
int pOpcja(void);
void OtworzOkna(void);
void ZamknijOkna(void);
void komunikat(char*);
void wyniki(void);
void pomoc(void);

```

```
class TApp : public TApplication
```

```
{
```

```
public:
```

```

    TApp();
    virtual void handleEvent( TEvent& event );
    static TMenuBar *initMenuBar( TRect );
    static TStatusLine *initStatusLine( TRect );
    virtual void getEvent(TEvent &event);
    void OknoAkcja();
    void OknoStosy();
    void OknoTasma();
    void OknoPrzejscia();
    void OknoAlterPrzejscia();

```

```

private:
    void greetingBox();
    void aboutDlgBox();
    void dosShell();
};
TApp automat;

class TDTp : public TDesktop
{
public:
    TDTp(const TRect& r);
    static TBackground *initBackground( TRect r );
};

TDTp::TDTp( const TRect& r ) : TDesktop(r),
    TDesktopInit( TDTp::initBackground)
{
}

TDesktopInit::TDesktopInit( TBackground *(*cBackground)( TRect ) ) :
    createBackground( cBackground )
{
}

TBackground *TDTp::initBackground(TRect r)
{
    return new TBackground(r,177);
}

TApp::TApp() :
    TProgInit( &TApp::initStatusLine,
               &TApp::initMenuBar,
               &TApp::initDesktop
               )
{
    katszlowa=0; // w1,...,w8
    trybPracy=0; // krokowa,ciagla
}

class TOknoW : public TWindow
{
public:
    TOknoW( const TRect& bounds, const char *aTitle, short aNumber );
};

class TOknoH : public TWindow
{
public:
    TOknoH( const TRect& bounds, const char *aTitle, short aNumber );
    virtual void handleEvent( TEvent& event );
};

void TOknoH::handleEvent( TEvent& event )
{
    TWindow::handleEvent(event);
}

class TOknoP : public TWindow
{
public:

```

```

        TOknoP( const TRect& bounds, const char *aTitle, short aNumber );
        virtual void handleEvent( TEvent& event );
};

class TOknoTW : public TWindow
{
public:
    TOknoTW( const TRect& bounds, const char *aTitle, short aNumber );
    virtual void handleEvent( TEvent& event );
};

class TOknoA : public TWindow
{
public:
    TOknoA( const TRect& bounds, const char *aTitle, short aNumber );
    virtual void handleEvent( TEvent& event );
};

class TWnetrze2 : public TView
{
public:
    TWnetrze2( const TRect& bounds );
    virtual void draw();
};

class TWnetrze3 : public TView
{
public:
    TWnetrze3( const TRect& bounds );
    virtual void draw();
};

class TOknoS : public TWindow
{
public:
    TOknoS( const TRect& r, const char *aTitle, short aNumber );
    virtual void handleEvent( TEvent& event );
};

class TOknoAP : public TWindow
{
public:
    TOknoAP( const TRect& r, const char *aTitle, short aNumber );
    virtual void handleEvent( TEvent& event );
};

class TWnetrze1 : public TScroller
{
public:
    TWnetrze1( const TRect& bounds, TScrollBar *aHScrollBar,
               TScrollBar *aVScrollBar );
    virtual void draw();
    virtual void handleEvent( TEvent& event );
};

class TWnetrzeS : public TScroller
{
public:
    TWnetrzeS( const TRect& bounds, TScrollBar *aHScrollBar,
               TScrollBar *aVScrollBar );

```

```

        virtual void draw();
};

class TWnetrzeAP : public TScroller
{
public:
    TWnetrzeAP( const TRect& bounds, TScrollBar *aHScrollBar,
                TScrollBar *aVScrollBar );
    virtual void draw();
};

class TWnetrze4 : public TScroller
{
public:
    TWnetrze4( const TRect& bounds, TScrollBar *aHScrollBar,
                TScrollBar *aVScrollBar );
    virtual void draw();
};

class TWnetrzeH : public TScroller
{
public:
    TWnetrzeH( const TRect& bounds, TScrollBar *aHScrollBar,
                TScrollBar *aVScrollBar );
    virtual void draw();
};

TWnetrze1::TWnetrze1( const TRect& bounds, TScrollBar *aHScrollBar,
                      TScrollBar *aVScrollBar ) :
    TScroller( bounds, aHScrollBar, aVScrollBar )
{
    growMode = gfGrowHiX | gfGrowHiY;
    options = options | ofFramed;
    setLimit( maxLineLength, maxLines );
}

TWnetrzeS::TWnetrzeS( const TRect& bounds, TScrollBar *aHScrollBar,
                      TScrollBar *aVScrollBar ) :
    TScroller( bounds, aHScrollBar, aVScrollBar )
{
    growMode = 0;
    options = options | ofFramed;
    setLimit( maxLineLength, maxLines );
}

TWnetrzeAP::TWnetrzeAP( const TRect& bounds, TScrollBar *aHScrollBar,
                        TScrollBar *aVScrollBar ) :
    TScroller( bounds, aHScrollBar, aVScrollBar )
{
    growMode = 0;
    options = options | ofFramed;
    setLimit( maxLineLength, maxLines );
}

TWnetrze4::TWnetrze4( const TRect& bounds, TScrollBar *aHScrollBar,
                      TScrollBar *aVScrollBar ) :
    TScroller( bounds, aHScrollBar, aVScrollBar )
{
    growMode = gfGrowHiX | gfGrowHiY;
    options = options | ofFramed;

```

```

        setLimit( maxLineLength, maxLines );
    }

    TWnetrzeH::TWnetrzeH( const TRect& bounds, TScrollBar *aHScrollBar,
                          TScrollBar *aVScrollBar ) :
        TScroller( bounds, aHScrollBar, aVScrollBar )
    {
        growMode = gfGrowHiX | gfGrowHiY;
        options = options | ofFramed;
        setLimit( maxLineLength, maxLines );
    }

    void TWnetrze1::draw()
    {
        ushort color = getColor(0x0302);

        for( int i = 0; i < size.y; i++ )
        {
            TDrawBuffer b;
            b.moveChar( 0, ' ', color, size.x );
            int j = delta.y + i;
            if( lines[j] )
            {
                char s[maxLineLength];
                if( delta.x > strlen(lines[j] ) )
                    s[0] = EOS;
                else
                {
                    strncpy( s, lines[j]+delta.x, size.x );
                    s[size.x] = EOS;
                }
                b.moveStr( 0, s, color );
            }
            writeLine( 0, i, size.x, 1, b);
        }
    }

    void TWnetrzeS::draw()
    {
        ushort color = getColor(0x0301);

        for( int i = 0; i < size.y; i++ )
        {
            TDrawBuffer b;
            b.moveChar( 0, ' ', color, size.x );
            int j = delta.y + i;
            if( plines[j] )
            {
                char s[maxLineLength];
                if( delta.x > strlen(plines[j] ) )
                    s[0] = EOS;
                else
                {
                    strncpy( s, plines[j]+delta.x, size.x );
                    s[size.x] = EOS;
                }
                b.moveStr( 0, s, color );
            }
            writeLine( 0, i, size.x, 1, b);
        }
    }

```

```

}

void TWnetrzeAP::draw()
{
    ushort color = getColor(0x0301);

    for( int i = 0; i < size.y; i++ )
    {
        TDrawBuffer b;
        b.moveChar( 0, ' ', color, size.x );
        int j = delta.y + i;
        if( aplines[j] )
        {
            char s[maxLineLength];
            if( delta.x > strlen(aplines[j] ) )
                s[0] = EOS;
            else
            {
                strncpy( s, aplines[j]+delta.x, size.x );
                s[size.x] = EOS;
            }
            b.moveStr( 0, s, color );
        }
        writeLine( 0, i, size.x, 1, b);
    }
}

```

```

void TWnetrze4::draw()
{
    ushort color = getColor(0x0301);

    for( int i = 0; i < size.y; i++ )
    {
        TDrawBuffer b;
        b.moveChar( 0, ' ', color, size.x );
        int j = delta.y + i;
        if( wlines[j] )
        {
            char s[maxLineLength];
            if( delta.x > strlen(wlines[j] ) )
                s[0] = EOS;
            else
            {
                strncpy( s, wlines[j]+delta.x, size.x );
                s[size.x] = EOS;
            }
            b.moveStr( 0, s, color );
        }
        writeLine( 0, i, size.x, 1, b);
    }
}

```

```

void TWnetrzeH::draw()
{
    ushort color = getColor(0x0301);

    for( int i = 0; i < size.y; i++ )
    {
        TDrawBuffer b;
        b.moveChar( 0, ' ', color, size.x );

```

```

        int j = delta.y + i;
        if( hlines[j] )
        {
            char s[maxLineLength];
            if( delta.x > strlen(hlines[j] ) )
                s[0] = EOS;
            else
            {
                strncpy( s, hlines[j]+delta.x, size.x );
                s[size.x] = EOS;
            }
            b.moveStr( 0, s, color );
        }
        writeLine( 0, i, size.x, 1, b);
    }
}

class KolekcjaSlow : public TStringCollection
{
public:
    KolekcjaSlow( short aLimit, short aDelta ) :
        TStringCollection( aLimit, aDelta ) {}
};

void PodzielNaSlowa(void);

KolekcjaSlow *kolekcja = new KolekcjaSlow( 20, 5 );

void TApp::greetingBox()
{
    TDialog *d = new TDialog( TRect( 5, 5, 75, 12 ), "Wprowadz zdanie" );

    d->insert( new TButton( TRect(4, 4, 25, 6) , " Akceptacja" , cmOK , bfDefault));
    d->insert( new TButton( TRect(40, 4, 65, 6), " Rezygnacja", cmCancel, bfNormal));
    d->insert( new TInputLine( TRect(2,2,68,3), 120) );
    d->options |= ofCentered;
    d->setData(zdanie);

    if( deskTop->execView( d ) != cmCancel )
    {
        d->getData(zdanie);
        ciagW[0]=EOS;
        PodzielNaSlowa();
        if( kolekcja->getCount() == 0 )
            komunikat("Nie wprowadzono zdania");
    }
}

void TApp::dosShell()
{
    suspend();
    system("cls");
    cout << "Napisz EXIT aby wrocic do programu...";
    system( getenv( "COMSPEC" ));
    resume();
    redraw();
}

void TApp::getEvent( TEvent& event )
{

```



```
static Boolean helpInUse = False;
```

```

TApplication::getEvent(event);
switch (event.what)
{
    case evCommand:
        if ((event.message.command == cmHelp) && ( helpInUse == False))
        {
            helpInUse = True;
            pomoc();
            helpInUse = False;
        }
        break;
    case evMouseDown:
        if (event.mouse.buttons != 1)
            event.what = evNothing;
        break;
}
}

```

```
void TApp::handleEvent( TEvent& event )
```

```

{
    ushort newMode;

    TApplication::handleEvent( event );
    if( event.what == evCommand )
    {
        switch( event.message.command )
        {
            case cmEdycja:
                greetingBox();
                clearEvent( event );
                break;
            case CzolowkaCmd:
                aboutDlgBox();
                break;
            case cmDosShell:
                dosShell();
                break;
            case cmVideoMode:
                newMode = TScreen::screenMode ^ TDisplay::smFont8x8;
                setScreenMode(newMode);
                break;
            case cmPodzial:
                for( int i = 0; i < maxLines; i++ )
                    { delete lines[i]; lines[i]=NULL; }
                lineCount = 0;
                OtworzOkna();
                pdams();
                break;
            case cmPOpcje:
                pOpcja();
                break;
            case cmPWyniki:
                wyniki();
                break;
            case cmOkno:
                OtworzOkna();
                break;
            case cmCloseAll:

```

```

        ZamknijOkna();
        break;
    default:
        return;
    }
    clearEvent(event);
}

int lkat(char *str)
{
    char s[120]="Podaj kategorie syntakt. slowa: ";
    TDialog *a = new TDialog (TRect(5, 5, 60, 18), strcat(s,str));
    TView *b = new TRadioButtons (TRect(2, 3, 25, 11),

        new TListItem( "Rzeczownik (w~1~)",
        new TListItem( "Zaimek (w~2~)",
        new TListItem( "Rodzajnik (w~3~)",
        new TListItem( "Przymiotnik (w~4~)",
        new TListItem( "Czasownik (w~5~)",
        new TListItem( "Przysłów (w~6~)",
        new TListItem( "Przyimek (w~7~)",
        new TListItem( "Spójnik (w~8~)", 0 )
        ))))));

    a->insert( new TLabel( TRect(2, 2, 25, 3),"Kategoria syntaktyczna",b));
    a->insert( new TButton( TRect(30, 4, 50, 6) , " Akceptacja" , cmOK , bfDefault));
    a->insert( new TButton( TRect(30, 8, 50, 10), " Rezygnacja", cmCancel, bfNormal));
    a->options |= ofCentered;
    a->insert(b);
    b->setData(&(katslowa=0));
    if ( TProgram::deskTop->execView(a) != cmCancel)
        b->getData(&katslowa);
    else
        return -1;
    return 0;
}

int pOpcja(void)
{
    TDialog *a = new TDialog (TRect(5, 5, 60, 12), "Wybierz tryb pracy");
    TView *b = new TRadioButtons (TRect(4, 3, 22, 5),
        new TListItem( "~K~rokowa",
        new TListItem( "~C~iagła", 0)
        ));

    a->insert( new TLabel( TRect(5, 2, 25, 3),"Tryb pracy",b));
    a->insert( new TButton( TRect(30, 2, 50, 4) , " Akceptacja" , cmOK , bfDefault));
    a->insert( new TButton( TRect(30, 4, 50, 6), " Rezygnacja", cmCancel, bfNormal));
    a->options |= ofCentered;
    a->insert(b);
    b->setData(&trybPracy);
    if ( TProgram::deskTop->execView(a) != cmCancel)
        b->getData(&trybPracy);
    else
        return -1;
    return 0;
}

```

```

void PodzielNaSlova(void)
{
    char *aktu,
        *start,
        wstr[80],
        ch = EOS;
    Boolean slowo;

    kolekcja->freeAll();
    for(int i=0;i<maxLines;i++) {delete slova[i];slova[i]=NULL; }; sCount=0;
    aktu = start = zdanie;
    slowo = False;

    while( (ch = *aktu) != EOS )
    {
        if( !isalnum( ch ) && slowo )
        {
            slowo = False;
            strncpy( wstr, start, size_t(aktu - start) );
            wstr[size_t(aktu-start)]=EOS;
            kolekcja->insert( newStr( strupr(wstr) ) );
            slova[sCount++]=newStr( strupr(wstr) );
            if(lkat(strupr(wstr)) == -1) return;
            strcat(ciagW.symbolW(katslova));
        }
        else if( isalnum( ch ) && !slowo )
        {
            slowo = True;
            start = aktu;
        }
        aktu++;
    }
    if( slowo )
    {
        strncpy( wstr, start, size_t(aktu - start) );
        wstr[size_t(aktu-start)]=EOS;
        kolekcja->insert( newStr( strupr(wstr) ) );
        slova[sCount++]=newStr( strupr(wstr) );
        if(lkat(strupr(wstr)) == -1) return;
        strcat(ciagW.symbolW(katslova));
    }
}

char *symbolW(ushort w)
{
    char *s;

    switch(w)
    {
        case 0: s="w1"; break;
        case 1: s="w2"; break;
        case 2: s="w3"; break;
        case 3: s="w4"; break;
        case 4: s="w5"; break;
        case 5: s="w6"; break;
        case 6: s="w7"; break;
        case 7: s="w8"; break;
    };
    return newStr(s);
}

```

```

TMenuBar *TApp::initMenuBar( TRect r )
{
    r.b.y = r.a.y+1;

    TMenuBar *a= new TMenuBar( r,
        *new TSubMenu( "~\360~", 0) +
            *new TMenuItem( "O~P~rogramie...", CzoLowkaCmd, kbNoKey ) +
            newLine() +
            *new TMenuItem( "~4~3/50 linii", cmVideoMode, kbAltG, hcNoContext, "Alt+G" )
+
            *new TMenuItem( "Poziom ~D~os'a", cmDosShell , kbNoKey ) +

        *new TSubMenu( "~Z~danie", kbAltZ ) +
            *new TMenuItem( "~E~dycja", cmEdycja, kbNoKey ) +

        *new TSubMenu( "~P~odzial", kbAltP )+
            *new TMenuItem( "~D~ziel", cmPodzial, kbNoKey )+
            *new TMenuItem( "~W~yniki", cmPWyniki, kbNoKey )+
            *new TMenuItem( "~O~pcje...", cmPOpcje, kbNoKey )+

        *new TSubMenu( "~O~kno", kbAltW )+
            *new TMenuItem( "~O~tworz wszystkie", cmOkno, kbF4, hcNoContext, "F4" )+
            newLine() +
            *new TMenuItem( "~N~astepne", cmNext, kbF6, hcNoContext, "F6" )+
            *new TMenuItem( "~P~owieksz", cmZoom, kbF5, hcNoContext, "F5" )+
            *new TMenuItem( "~Z~amknij", cmClose, kbAltF3, hcNoContext, "Alt+F3" )+
            *new TMenuItem( "Zamknij ~w~szystkie", cmCloseAll, kbNoKey )+

        *new TSubMenu( "~W~yjście", kbAltW ) +
            *new TMenuItem( "~W~yjście", cmQuit, cmQuit, hcNoContext, "Alt-W" )
    );
    return a;
}

TStatusLine *TApp::initStatusLine( TRect r )
{
    r.a.y = r.b.y-1;

    return new TStatusLine( r,
        *new TStatusDef( 0, 0xFFFF ) +
            *new TStatusItem( "~F1~ Pomoc", kbF1, cmHelp ) +
            *new TStatusItem( "~F10~ Menu", kbF10, cmMenu ) +
            *new TStatusItem( "~Alt-W~ Wyjście", kbAltW, cmQuit )
    );
}

void TApp::aboutDlgBox()
{
    TDialog *aboutBox = new TDialog(TRect(0, 0, 49, 13), "Kilka slow o programie");

    aboutBox->insert
    (
        new TStaticText(TRect(9, 2, 40, 9),
            "\003Komputerowa realizacja"
            "\n\003automatu P D A M S\n\003\n"
            "\003      \n"
            "\n\003Copyright (c) 1993"
            "\n\003Olgerd Unold"
        )
    );
}

```

```

        aboutBox->insert( new TButton(TRect(20, 10, 30, 12), "OK", cmOK, bfDefault) );
        aboutBox->options |= ofCentered;
        deskTop->execView(aboutBox);
        destroy( aboutBox );
    }

void komunikat(char *str)
{
    TDialog *a = new TDialog(TRect(0, 0, strlen(str)+4, 7),"komunikat" );

    a->insert( new TStaticText(TRect(2, 2, strlen(str)+2, 3),str));
    a->insert( new TButton(TRect(strlen(str)/2-1, 4, strlen(str)/2+5, 6), "OK", cmOK, bfDefault) );

    a->options |= ofCentered;
    TProgram::deskTop->execView(a);
    TObject::destroy( a );
}

static short winNumer = 0;

TOknoS::TOknoS( const TRect& bounds, const char *aTitle,
                short aNumber ) :
    TWindow( bounds, aTitle, aNumber),
    TWindowInit( &TOknoS::initFrame )
{
    TScrollBar *hScrollBar=standardScrollBar( sbHorizontal | sbHandleKeyboard );

    growMode = 0;
    flags = wfMove | wfClose;
    options |= ofPostProcess;
    TRect r = getClipRect();
    r.grow(-1, -1);
    insert( new TWnetrzeS(r,hScrollBar,0));
}

TOknoAP::TOknoAP( const TRect& bounds, const char *aTitle,
                  short aNumber ) :
    TWindow( bounds, aTitle, aNumber),
    TWindowInit( &TOknoS::initFrame )
{
    TScrollBar *vScrollBar =
        standardScrollBar( sbVertical | sbHandleKeyboard );

    growMode = 0;
    flags = wfMove | wfClose;
    options |= ofPostProcess;
    TRect r = getClipRect();
    r.grow(-1, -1);
    insert( new TWnetrzeAP(r,0,vScrollBar));
}

TOknoTW::TOknoTW( const TRect& bounds, const char *aTitle,
                  short aNumber ) :
    TWindow( bounds, aTitle, aNumber),
    TWindowInit( &TOknoS::initFrame )
{
    growMode = 0;
    flags = wfMove | wfClose;
    options |= ofPostProcess;

```

```

    TRect r = getClipRect();
    r.grow(-1, -1);
    insert( new TWnetrze2(r));
}

TOknoA::TOknoA( const TRect& bounds, const char *aTitle,
                short aNumber ) :
    TWindow( bounds, aTitle, aNumber),
    TWindowInit( &TOknoS::initFrame )
{
    growMode = 0;
    flags = wfMove | wfClose;
    options |= ofPostProcess;
    TRect r = getClipRect();
    r.grow(-1, -1);
    insert( new TWnetrze3(r));
}

TOknoP::TOknoP( const TRect& bounds, const char *aTitle,
                short aNumber ) :
    TWindow( bounds, aTitle, aNumber),
    TWindowInit( &TOknoS::initFrame )
{
    TScrollBar *vScrollBar =
        standardScrollBar( sbVertical | sbHandleKeyboard );
    TScrollBar *hScrollBar =
        standardScrollBar( sbHorizontal | sbHandleKeyboard );

    options |= ofPostProcess;
    TRect r = getClipRect();
    r.grow(-1, -1);
    insert( new TWnetrze1(r,hScrollBar,vScrollBar));
}

TOknoW::TOknoW( const TRect& bounds, const char *aTitle,
                short aNumber ) :
    TWindow( bounds, aTitle, aNumber),
    TWindowInit( &TOknoS::initFrame )
{
    for( int i = 0; i < maxLines; i++ ) { delete wlines[i]; wlines[i]=NULL; } wlineCount=0;
    wlines[wlineCount++] = new Str("ZDANIE WEJSCIOWE (s):");
    wlines[wlineCount++] = new Str(zdanie);
    wlines[wlineCount++] = new Str("");
    wlines[wlineCount++] = new Str("SKOJARZONY ZE ZDANIEM (s) CIAG SYMBOLI (w):");
    wlines[wlineCount++] = new Str(ciagW);
    wlines[wlineCount++] = new Str("");
    podzialW();

    TScrollBar *vScrollBar =
        standardScrollBar( sbVertical | sbHandleKeyboard );
    TScrollBar *hScrollBar =
        standardScrollBar( sbHorizontal | sbHandleKeyboard );
    options |= ofPostProcess | ofCentered;
    TRect r = getClipRect();
    r.grow(-1, -1);
    insert( new TWnetrze4(r,hScrollBar,vScrollBar));
}

TOknoH::TOknoH( const TRect& bounds, const char *aTitle,
                short aNumber ) :

```

```

        TWindow( bounds, aTitle, aNumber),
        TWindowInit( &TOknoS::initFrame )
    {
        TScrollBar *vScrollBar =
            standardScrollBar( sbVertical | sbHandleKeyboard );
        TScrollBar *hScrollBar =
            standardScrollBar( sbHorizontal | sbHandleKeyboard );

        options |= ofPostProcess | ofCentered;
        TRect r = getClipRect();
        r.grow(-1, -1);
        insert( new TWnetrzeH(r,hScrollBar,vScrollBar));
    }

void TWnetrze1::handleEvent( TEvent& event )
{
    TPoint d=delta;

    TScroller::handleEvent(event);
    if( event.what == evBroadcast && event.message.command==cmPiszwOkno)
        if(size.y < lineCount)
            {d.y++; scrollTo(d.x, d.y);}
}

void TOknoP::handleEvent( TEvent& event )
{
    TWindow::handleEvent( event );
    if( event.what == evBroadcast)
    {
        switch( event.message.command )
        {
            case cmPiszwOkno:
                hide();
                show();
                break;
            case cmCloseAll:
                close();
                break;
            default:
                return;
        };
    }
}

void TOknoTW::handleEvent( TEvent& event )
{
    TWindow::handleEvent( event );
    if( event.what == evBroadcast)
    {
        switch( event.message.command )
        {
            case cmPiszwOkno:
                hide();
                show();
                break;
            case cmCloseAll:
                close();
                break;
            default:
                return;
        };
    }
}

```

```

        return;
    };
}

void TOKnoA::handleEvent( TEvent& event )
{
    TWindow::handleEvent( event );
    if( event.what == evBroadcast )
    {
        switch( event.message.command )
        {
            case cmPiszwOkno:
                hide();
                show();
                break;
            case cmCloseAll:
                close();
                break;
            default:
                return;
        };
    }
}

```

```

void TOKnoS::handleEvent( TEvent& event )
{
    ushort newMode;

    TWindow::handleEvent( event );
    if( event.what == evBroadcast )
    {
        switch( event.message.command )
        {
            case cmPiszwOkno:
                hide();
                show();
                break;
            case cmCloseAll:
                close();
                break;
            default:
                return;
        };
    }
}

```

```

void TOKnoAP::handleEvent( TEvent& event )
{
    TWindow::handleEvent( event );
    if( event.what == evBroadcast )
    {
        switch( event.message.command )
        {
            case cmPiszwOkno:
                hide();
                show();
                break;
            case cmCloseAll:
                close();

```



```

        break;
    default:
        return;
    };
}

TWnetrze2::TWnetrze2( const TRect& bounds ) : TView( bounds )
{
    growMode = 0;
    options = options | offFramed;
}

void TWnetrze2::draw()
{
    ushort color = getColor(0x0302);
    for( int i = 0; i < size.y; i++ )
    {
        TDrawBuffer b;
        b.moveChar( 0, ' ', color, size.x );
        if( tlines[i] )
        {
            char s[maxLineLength];
            strncpy( s, tlines[i], size.x );
            s[size.x] = EOS;
            b.moveStr( 0, s, color );
        }
        writeLine( 0, i, size.x, 1, b);
    }
}

TWnetrze3::TWnetrze3( const TRect& bounds ) : TView( bounds )
{
    growMode = 0;
    options = options | offFramed;
}

void TWnetrze3::draw()
{
    ushort color = getColor(0x0301);

    for( int i = 0; i < size.y; i++ )
    {
        TDrawBuffer b;
        b.moveChar( 0, ' ', color, size.x );
        if( alines[i] )
        {
            char s[maxLineLength];
            strncpy( s, alines[i], size.x );
            s[size.x] = EOS;
            b.moveStr( 0, s, color );
        }
        writeLine( 0, i, size.x, 1, b);
    }
}

void TApp::OknoPrzejscia()
{
    TRect r( 0, 0, 50, 9 );
    r.move(1, 14 );
    TOknoP *window = new TOknoP ( r, "Przejscia ze stanu do stanu", ++winNumer);
}

```

```

        deskTop->insert(window);
    }

void TApp::OknoAlterPrzejscia()
{
    TRect r( 0, 0, 20, 10 );
    r.move( 55, 11 );
    TOknoAP *window = new TOknoAP ( r, "Alter.", ++winNumer);
    deskTop->insert(window);
}

void TApp::OknoTasma()
{
    TRect r( 0, 0, 60, 4 ); r.move( 1, 0 );
    TOknoTW *window = new TOknoTW ( r, "Tasma Wejsckowa", ++winNumer);
    deskTop->insert(window);
}

void TApp::OknoStosy()
{
    TRect r( 0, 0, 50, 8 ); r.move( 1, 5 );
    TOknoS *window = new TOknoS ( r, "Zawartosc Stosow", ++winNumer);
    deskTop->insert(window);
}

void TApp::OknoAkcja()
{
    TRect r( 0, 0, 20, 4 ); r.move( 55, 5 );
    TOknoA *window = new TOknoA ( r, "Instrukcja", ++winNumer);
    deskTop->insert(window);
}

void OtworzOkna(void)
{
    automat.OknoAlterPrzejscia();
    automat.OknoAkcja();
    automat.OknoStosy();
    automat.OknoTasma();
    automat.OknoPrzejscia();
}

void ZamknijOkna(void)
{
    message(TProgram::deskTop, evBroadcast, cmCloseAll, 0);
    winNumer=0;
}

void wyniki(void)
{
    TProgram::deskTop->insert( new TOknoW ( TRect( 0, 0, 60, 16 ), "Wynik podzialu zdania
na komponenty strukt-sem", ++winNumer));
}
TOknoH *OknoPomoc=new TOknoH ( TRect( 0, 0, 60, 16 ), "Pomoc", ++winNumer);
void pomoc(void)
{
    TProgram::deskTop->insert( OknoPomoc);
}

```

```

void main(void)
{
    hlines[hlineCount++]=newStr(" Program");
    hlines[hlineCount++]=newStr("jest przykladem komputerowej realizacji automatu
PDAMS");
    hlines[hlineCount++]=newStr("dokonujacego podzialu ciagu (w) skojarzonego ze zdaniem");
    hlines[hlineCount++]=newStr("jezyka angielskiego na podciagi.");
    hlines[hlineCount++]=newStr("Kazdy symbol 'wi' z ciagu (w) okresla kategorie
leksykalna");
    hlines[hlineCount++]=newStr("odpowiadajacego symbolowi slowa w zdaniu.");
    hlines[hlineCount++]=newStr("");
    hlines[hlineCount++]=newStr(" Dzieki jednoznaczniemu przyporzadkowaniu kazdego
symbolu");
    hlines[hlineCount++]=newStr("'wi' z ciagu (w) do poszczegolnych slow zdania.");
    hlines[hlineCount++]=newStr("mozna odwzorowac otrzymane podciagi ciagu (w)");
    hlines[hlineCount++]=newStr("na grupy slow.");
    hlines[hlineCount++]=newStr("");
    hlines[hlineCount++]=newStr(" Podczas podzialu ciagu (w) na podciagi, automat okresla");
    hlines[hlineCount++]=newStr("kategorie syntaktyczna 'li' podciagu i w konsekwencji");
    hlines[hlineCount++]=newStr("kategorie syntaktyczna odpowiadajacej podciagowi ");
    hlines[hlineCount++]=newStr("grupy slow.");
    hlines[hlineCount++]=newStr("");
    hlines[hlineCount++]=newStr(" Objasnienie stosowanych symboli 'wi': ('wi' nalezy do zbioru
W)");
    hlines[hlineCount++]=newStr("w1 - rzeczownik.");
    hlines[hlineCount++]=newStr("w2 - zaimek.");
    hlines[hlineCount++]=newStr("w3 - rodzajnik.");
    hlines[hlineCount++]=newStr("w4 - przymiotnik.");
    hlines[hlineCount++]=newStr("w5 - czasownik.");
    hlines[hlineCount++]=newStr("w6 - przyslowek.");
    hlines[hlineCount++]=newStr("w7 - przyimek.");
    hlines[hlineCount++]=newStr("w8 - spojnik.");
    hlines[hlineCount++]=newStr("w9 - liczebnik.");
    hlines[hlineCount++]=newStr("");
    hlines[hlineCount++]=newStr(" Objasnienie stosowanych symboli 'li':");
    hlines[hlineCount++]=newStr("l1 - grupa rzeczownikowa.");
    hlines[hlineCount++]=newStr("l2 -      czasownikowa.");
    hlines[hlineCount++]=newStr("l3 -      przymiotnikowa.");
    hlines[hlineCount++]=newStr("l4 -      przyslowkowa.");
    hlines[hlineCount++]=newStr("l5 -      przyimkowa.");
    hlines[hlineCount++]=newStr("l6 -      czasownikowa nieosobowa.");
    hlines[hlineCount++]=newStr("");
    hlines[hlineCount++]=newStr(" Podzial ciagu (w) na podciagi jest realizowany przez ");
    hlines[hlineCount++]=newStr("automat ze stosami i skojarzeniowym dostepem do");
    hlines[hlineCount++]=newStr("pamieci (ang. Pushdown Automata with Associative Memory
Acces - PDAMS)");
    hlines[hlineCount++]=newStr("Automat podczas dzialania operuje na szesciu stosach");
    hlines[hlineCount++]=newStr("(S1,...,S6).");
    hlines[hlineCount++]=newStr("Automat moze wykonac jedna z siedmiu instrukcji: ");
    hlines[hlineCount++]=newStr("1) instrukcja i1");
    hlines[hlineCount++]=newStr(" wstawia na S3 czytany z tasmy wejsciowej (TW)");
    hlines[hlineCount++]=newStr(" symbol 'wi' i przesuwa glowice o jedna klatke w prawo.");
    hlines[hlineCount++]=newStr("2) instrukcja i2");
    hlines[hlineCount++]=newStr(" przepisuje grupe symboli ze zbioru W zapisana na S6.");
    hlines[hlineCount++]=newStr(" liczac od szczytu stosu do pierwszego symbolu 'li'.");
    hlines[hlineCount++]=newStr(" oraz ciagu nastepujacych po sobie symboli ze zbioru L");
    hlines[hlineCount++]=newStr(" na stos S3.");
    hlines[hlineCount++]=newStr("3) instrukcja i3");
    hlines[hlineCount++]=newStr(" zapamieta aktualna konfiguracje automatu na stosach.");
    hlines[hlineCount++]=newStr("4) instrukcja i4");

```


ZALĄCZNIK B

Analizę gramatyczną komponentu o strukturze grupy werbalnej osobowej (GWO) ułatwia przeglądanie komponentu od końca, tzn. od ostatniego czasownika komponentu (czasownika głównego) do jego pierwszego czasownika (operatora). Przykładowo, kolejność analizowanych czasowników wchodzących w skład grupy *will have been taken* będzie następująca: *taken, been, have, will*.

Procedura *Analizy* korzysta ze zmiennej - *word*, pięciu funkcji - *next, root, form, eog, error*, pięciu procedur - *Not, Are, Mod, Perf, DontMod* oraz dwóch zbiorów - *modal_in_present, modal_in_past*:

word - zmienna zawierająca aktualnie przeglądane słowo grupy,

next - funkcja przypisująca zmiennej *word* kolejne słowo grupy (kolejne licząc od końca grupy),

root - funkcja zwracająca formę podstawową (bezokolicznik) czasownika ze zmiennej *word*,

form - funkcja rozpoznająca formę czasownika ze zmiennej *word*; funkcja *form* zwraca jedną z czterech wartości:

ING - czasownik ze zmiennej *word* jest w formie present participle,

ED - czasownik ze zmiennej *word* jest w formie past tense,

EN - czasownik ze zmiennej *word* jest w formie past participle,

FROOT - czasownik ze zmiennej *word* jest w formie podstawowej,

eog - funkcja rozpoznająca koniec grupy,

error - funkcja przerywająca analizę gramatyczną grupy w momencie rozpoznania błędu syntaktycznego,

Not - procedura rozpoznająca partykulę przeczącą 'not',

Are - procedura rozpoznająca czasownik 'be' w czasie gramatycznym simple present ('is', 'am', 'are') lub w czasie simple past ('was', 'were'),

Mod - procedura rozpoznająca czasownik modalny w czasie gramatycznym present lub past,

Perf - procedura rozpoznająca konstrukcję składniową typu perfect; konstrukcja może być poprzedzona czasownikiem modalnym w czasie gramatycznym present lub past,

DontMod - procedura rozpoznająca czasownik specjalny 'do' lub czasownik modalny; oba czasowniki mogą być w czasie gramatycznym present lub past,

modal_in_present - zbiór czasowników modalnych w czasie gramatycznym present {can, may, must, will, shall, need, dare},

modal_in_past - zbiór czasowników modalnych w czasie gramatycznym past {could, might, would, should}.

```

procedure Not
begin
  if word = 'not' then
    begin
      NEGATION:=YES;
    end
  end
end

```

```

procedure Are
begin
  Not;
  if word = 'am' or word = 'are' or word = 'is' then;
  else
    if word = 'was' or word = 'were' then
      PAST:=YES;
    else error
  end
end

```

```

procedure Mod
begin
  Not;
  if word in modal_in_present then
    MODALITY:=word
  else
    if word in modal_in_past then
      begin
        MODALITY:=root;
        PAST:=YES
      end
    else error
  end
end

```

```

procedure Perf
begin
  Not;
  if word = 'had' then
    begin
      PERFECT:=YES;
      PAST:=YES
    end
  else
    if word = 'has' then
      PERFECT:=YES
    else
      if word = 'have' then
        begin

```

```

    PERFECT:=YES;
    next;
    if eog then;
    else Mod
    end
    else error;
end

```

```

procedure DontMod
begin
    if word = 'do' or word = 'does' then;
    else
        if word = 'did' then
            PAST:=YES;
        else Mod
        end
    end
end

```

```

procedure Analyzer
begin
    PAST:=NOT; CONTINUOUS:=NOT; PERFECT:=NOT; PASSIVE:=NOT;
    MODALITY:=""; MAIN_VERB:="";
    if form = ING then
        begin
            MAIN_VERB:=root;
            CONTINUOUS:=YES;
            next;
            if word = 'be' then
                begin
                    next;
                    Mod
                end
            else
                if word = 'been' then
                    begin
                        next;
                        Perf
                    end
                else Are
                end
            end
        else
            if form = EN then
                begin
                    MAIN_VERB:=root;
                    next;
                    if word = 'been' then
                        begin
                            PASSIV:=YES;
                            next;

```

```

    Perf
  end
else
  begin
    Not;
    if word = 'has' or word = 'have' or word = 'had' then
      Perf
    else
      begin
        PASSIV:=YES;
        Are
      end
    end
  end
else
  if form = ED then
    begin
      MAIN_VERB:=root;
      PAST:=YES
    end
  else
    if form = FROOT then
      begin
        MAIN_VERB:=word;
        next;
        if eog then;
        else
          begin
            Not;
            DontMod
          end
        end
      end
    else error
  end
end

```


ZALĄCZNIK C

Procedura **Syntheser** korzysta z trzech zmiennych - *fvg*, *str_not*, *str_do* oraz czterech funkcji - *get_ing*, *get_en*, *get_ed*, *get_modal_past*:

fvg - zmienna tekstowa zawierająca odtworzony komponent; czasownik otoczony parą nawiasów kwadratowych wymaga jeszcze procesu uzgodnienia swojej formy osobowej,

str_not - zmienna tekstowa zawierająca w przypadku komponentu z negacją napis 'not'; w przeciwnym wypadku zmienna *not* zawiera napis pusty "",

str_do - zmienna tekstowa zawierająca w przypadku komponentu z negacją napis '[do]'; w przeciwnym wypadku zmienna *do* zawiera napis pusty "",

get_ing - funkcja zwracająca formę present participle czasownika ze zmiennej MAIN_VERB,

get_en - funkcja zwracająca formę past participle czasownika ze zmiennej MAIN_VERB,

get_ed - funkcja zwracająca formę past czasownika ze zmiennej MAIN_VERB,

get_modal_past - funkcja zwracająca formę past czasownika ze zmiennej MODALITY.

procedure **Syntheser**

begin

if NEGATION = YES then

begin

str_not:='not';

str_do:='[do]'

end

else

begin

str_not:="";

str_do:=""

end

if CONTINUOUS = YES then

begin

if PERFECT = YES then

begin

if MODALITY <> "" then

begin

if PAST = YES then

fvg:=get_modal_past+str_not+'have been'+get_ing

else

fvg:=MODALITY+str_not+'have been'+get_ing

end

else

```

if PAST = YES then
  fvg:='had'+str_not+'been'+get_ing
else
  fvg:='[have]'+str_not+'been'+get_ing
end
else
if MODALITY <> "" then
begin
  if PAST = YES then
    fvg:=get_modal_past+str_not+'be'+get_ing
  else
    fvg:=MODALITY+str_not+'be'+get_ing
  end
else
  if PAST = YES then
    fvg:='[were]'+str_not+get_ing
  else
    fvg:='[are]'+str_not+get_ing
  end
if PASSIVE = YES then
begin
  if PERFECT = YES then
begin
  if MODALITY <> "" then
begin
  if PAST = YES then
    fvg:=get_modal_past+str_not+'have been'+get_en
  else
    fvg:=MODALITY+str_not+'have been'+get_en
  end
else
  if PAST = YES then
    fvg:='had'+str_not+'been'+get_en
  else
    fvg:='[have]'+str_not+'been'+get_en
  end
else
  if PAST = YES then
    fvg:='[were]'+str_not+get_en
  else
    fvg:='[are]'+str_not+get_en
  end
else
if PERFECT = YES then
begin
  if MODALITY <> "" then
begin
  if PAST = YES then
    fvg:=get_modal_past+str_not+'have'+get_en
  else

```

```
    fvg:=MODALITY+str_not+'have'+get_en
  end
else
  if PAST = YES then
    fvg:='had'+str_not+get_en
  else
    fvg:='[have]'+str_not+get_en
  end
else
  if MODALITY <> "" then
    begin
      if PAST = YES then
        fvg:=get_modal_past+str_not+MAIN_VERB
      else
        fvg:=MODALITY+str_not+MAIN_VERB
      end
    end
  else
    if PAST = YES then
      begin
        if NEGATION = YES then
          fvg:='did not'+MAIN_VERB
        else
          fvg:=get_ed
        end
      end
    else
      fvg:=str_do+str_not+MAIN_VERB
    end
  end
end
```

mgr inż. Olgierd Unold
Instytut Cybernetyki Technicznej
Politechniki Wrocławskiej
ul Janiszewskiego 11/17
50-372 Wrocław

Niniejszy raport otrzymują:

1. Biblioteka Główna Politechniki Wrocławskiej	1 egz.
2. OINT I-6	1 egz.
3. Z-ca Dyrektora Instytutu d/s Rozwoju Kadry Naukowej	1 egz.
4. Promotor	1 egz.
5. Recenzenci	3 egz.
6. Autor	3 egz.

Razem: 10 egz.

Raport wpłynął do Redakcji I-6
w grudniu 1993 r.