

Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Łukasz Degórski

Nr albumu: 174785

Wykorzystanie analizy morfologicznej do obsługi korpusów

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dra hab. Janusza S. Bienia, prof. UW
Katedra Lingwistyki Formalnej
Wydział Neofilologii UW

Warszawa, wrzesień 2004

Oświadczenie kierującego pracą

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data

Podpis kierującego pracą

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przez mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data

Podpis autora pracy

Streszczenie

W ramach pracy przeprowadzony został przegląd korzystających z analizy morfologicznej narzędzi stosowanych do badania korpusów tekstów języka naturalnego, oraz stworzony przykładowy program tego typu.

Sporządzając odpowiednie konkordancje oprogramowanie takie może wspomagać pracę językoznawcy lub tłumacza. W bardziej zaawansowanych narzędziach zadanie do wykonania może zostać wyspecyfikowane nie tylko przez podanie wyszukiwanego ciągu znaków, ale też na przykład przez podanie formy hasłowej lub żądanej formy gramatycznej poszukiwanych wyrazów.

W pracy starałem się odnieść do wszystkich funkcji, jakie miewają narzędzia tego typu. W programie wprowadziłem też nowe, dotychczas nie spotykane – możliwość użycia w zapytaniach zmiennych, które zostaną uzgodnione w toku wyszukiwania, oraz wyszukiwanie w wynikach.

Słowa kluczowe

analiza języka naturalnego, analiza morfologiczna, konkordancja, korpus

Dziedzina pracy (kody wg programu Socrates-Erazmus)

11.3 Computer Science

Klasyfikacja tematyczna

I. Computing Methodologies

I.2. Artificial Intelligence

I.2.7. Natural Language Processing

Copyright (c) 2004 Łukasz Degórski

Zezwala się na kopiowanie, rozpowszechnianie i/lub modyfikowanie tego dokumentu na warunkach Licencji GNU Wolnej Dokumentacji (GNU Free Documentation License) w wersji 1.1 lub dowolnej nowszej opublikowanej przez Fundację Wolnego Oprogramowania, bez żadnych Części nienaruszalnych, bez Tekstów przedniej lub tylnej strony okładki. Egzemplarz licencji zamieszczono w części zatytułowanej "GNU Free Documentation License".

Nieoficjalne polskie tłumaczenie tekstu licencji jest dostępne pod adresem <http://www.gnu.org.pl/text/GFDL-pl.html>

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Spis treści

Wprowadzenie	5
I Istniejące narzędzia	9
1. Funkcje dostępne w programach tworzących konkordancje i ich znaczenie	11
1.1. Niezależne od struktury i zawartości korpusu	11
1.1.1. Możliwość zadawania zapytań wielowyrazowych	11
1.1.2. Możliwość użycia wyrażeń regularnych	11
1.1.3. Możliwość wyszukiwania wyrazów występujących w zadanej odległości od siebie	12
1.1.4. Możliwość sortowania wyników	13
1.1.5. Możliwość wyszukiwania w wynikach	13
1.2. Zależne od struktury i zawartości korpusu	13
1.2.1. Możliwość odwołania do formy hasłowej	14
1.2.2. Możliwość odwołania do współrzędnych morfologicznych	14
1.2.3. Możliwość odwołania do znaków interpunkcyjnych	14
1.2.4. Możliwość odwołania do metadanych	15
1.2.5. Możliwość wykorzystania uzgadniających się zmiennych	15
2. Przegląd programów tworzących konkordancje	17
2.1. IMS Corpus Workbench i CQP	17
2.2. Czeski Korpus Narodowy – interfejs dostępny przez WWW	18
2.3. Czeski Korpus Narodowy – program łączący się z serwerem Korpusu za pomocą protokołu Telnet	18
2.4. Węgierski Korpus Narodowy – ogólnodostępny interfejs WWW	19
2.5. Słowacki Korpus Narodowy – ogólnodostępny interfejs WWW	19
2.6. Korpus języka słoweńskiego Instytutu Frana Ramovša	20
2.7. Korpus języka słoweńskiego FIDA	20
2.8. Chorwacki Korpus Narodowy – ogólnodostępny interfejs WWW	21
2.9. The Oslo Corpus of Bosnian Texts – interfejs WWW	21
2.10. Korpus PWN – próbka dostępna przez WWW	21
2.11. Korpus PWN – próbka dostępna na płycie CD	22
2.12. Korpus IPI PAN – wersja dostępna na CD	22
2.13. Korpus IPI PAN – próbka dostępna przez WWW	23
2.14. British National Corpus – próbka dostępna przez WWW	23
3. Podsumowanie możliwości analizowanych programów	25
3.1. Przegląd	25
3.2. Wnioski	28

II Pescador – przykładowy system	29
4. Założenia wstępne	31
5. Wybór języka i narzędzi programistycznych	33
5.1. Język programowania	33
5.2. Kompilator	33
6. Schemat ogólny systemu	35
7. Formaty danych wejściowych	37
7.1. Format pliku zawierającego tekst korpusu	37
7.2. Format wewnętrzny anotacji	38
7.2.1. Problem niejednoznaczności segmentacji	39
7.2.2. Reprezentacja w pamięci podczas pracy programu	40
7.2.3. Reprezentacja w plikach roboczych	40
8. Język zapytań	43
8.1. Definicja jednego wyrazu	43
8.2. Predykaty specjalne	44
8.3. Gramatyka języka zapytań	44
8.4. Obsługa zmiennych	44
8.5. Traktowanie znaków interpunkcyjnych	45
8.6. Traktowanie podziałów wierszy	45
8.7. Przykłady zapytań	45
9. Formaty wyjściowe i eksport	47
10. Dokumentacja użytkowa	49
10.1. Uruchomienie serwera	49
10.2. Obsługa interfejsu	49
11. Konwertery wejściowe	51
11.1. Brak anotacji	51
11.2. SAM	51
12. Dokumentacja techniczna	53
12.1. Struktura	53
12.1.1. Serwer	53
12.1.2. Klient – przykładowy interfejs tekstowy	54
12.1.3. Konwerter informacji morfologicznej z analizatora SAM	55
12.1.4. Konwerter zerowej informacji morfologicznej	55
12.2. Niektóre algorytmy	55
12.2.1. Wyszukiwanie	55
12.2.2. Uzgadnianie zmiennych	58
12.2.3. Wyszukiwanie w wynikach	59
13. Możliwości dalszego rozwoju systemu	61
A. Opis dołączonej płyty z oprogramowaniem	63

B. Korespondencja z Radą Języka Polskiego	65
C. GNU Free Documentation License	67
D. Licencja na używanie modułów pochodzących z NJ SML	73

Wprowadzenie

W pierwszej części pracy prezentuję przegląd możliwości istniejących narzędzi korpusowych wykorzystujących analizę morfologiczną, poparty krótkimi opisami badanych programów i zbiorczym zestawieniem.

W drugiej części przedstawiam własny program Pescador, zawierający kilka funkcji, jakich nie udostępniają znane mi narzędzia istniejące. Dodatkowo program został napisany tak, by nie uzależniać się od budowy konkretnego języka. Powinien równie dobrze posłużyć do analizy danych z korpusu języka polskiego, jak i swahili, pod warunkiem oczywiście stworzenia modułów zewnętrznych obsługujących komunikację z korpusem i odpowiednim analizatorem morfologicznym. Stworzyłem taki moduł współpracujący z analizatorem języka polskiego SAM dr Krzysztofa Szafrana oraz moduł pozwalający analizować tekst nieanotowany. Dzięki temu ostatecznie, na poziomie badania zewnętrznych reprezentacji wyrazów (bez zapytań o współrzędne morfologiczne), Pescador może zostać bez żadnych dodatkowych narzędzi użyty na danych wejściowych w postaci pliku tekstowego.

W pierwotnym zamierzeniu trzecią częścią pracy miał być nowy analizator morfologiczny dla języka polskiego, oparty na indeksie Tokarskiego [7], który miał stać się następcą analizatora SAM. W czasie jej pisania został jednak ukończony w IPI PAN analizator *Morfeusz* Marcina Wolińskiego, również oparty na Indeksie. W związku z tym nie było sensu wykonywać równoległe tej samej pracy i mój analizator nie powstał. Niestety, ostateczny format wyjścia z *Morfeusza* nie był dostępny dostatecznie wcześniej, by stworzyć dla niego interfejs (konwerter wejściowy).

Poniżej wyjaśniam niektóre wykorzystywane w pracy terminy.

Analiza morfologiczna – badanie cech poszczególnych wyrazów z osobna na podstawie dostępnej reprezentacji zewnętrznej w postaci ciągu znaków. Wynikiem analizy jest zestaw współrzędnych morfologicznych badanego wyrazu, takich jak *liczba – pojedyncza, rodzaj – męski, stopień – wyższy, ...* Z racji występowania synkretyzmu form jedna zewnętrzna reprezentacja może mieć wiele różnych interpretacji, na przykład *komputer* może być zarówno mianownikiem, jak i biernikiem liczby pojedynczej. Rozróżnienie tych dwóch wartości kategorii przypadku nie leży już w zakresie możliwości analizy morfologicznej, lecz syntaktycznej, zajmującej się wzajemnymi zależnościami wyrazów w wypowiedzeniu: wiemy, że *komputer* w funkcji podmiotu występuje w mianowniku, a w funkcji dopełnienia - w bierniku. Warto dodać, że istnieją też sytuacje, w których nawet analiza syntaktyczna będzie bezradna; przykłady podaje Przepiórkowski w [5].

Kasza – górna lub dolna – zapis znaku wielkimi lub małymi literami.

Konkordancja – za [4]: zestawienie wszystkich słów z danego dzieła z odesłaniem do miejsc ich występowania w tekście. Konkordancje Biblii zaczęto zestawiać już od XIII w., ale pierwsza konkordancja tekstu niebiblijnego (dzieł Szekspira) powstała dopiero w 1787 roku ([4]). Nawet i dziś znakomita większość trafień w wyszukiwarce Google dla zapytania “konkordancja” odnosi się do konkordancji Biblii. Zdarzają się też opinie, że definicją konkordancji jest zestawienie słów konkretnie z Biblii.

Dopiero komputery i elektroniczny zapis tekstu umożliwiły poważne wykorzystanie konkordancji innych tekstów na szerszą skalę, bowiem ich ręczne sporządzanie wymaga zbyt wielkiego nakładu pracy, by wysiłek się opłacał.

Programy tworzące konkordancje bywają potocznie nazywane *konkordancerami* lub *konkordanserami*. Z uwagi jednak na stanowisko Rady Języka Polskiego (a raczej jego brak - korespondencję elektroniczną z Radą przytaczam w dodatku B) nie używam żadnej z tych form w pracy, posługując się terminami opisowymi.

Korpus języka naturalnego – zorganizowany zbiór tekstów języka naturalnego, ewentualnie wzbogacony o informacje gramatyczne dotyczące poszczególnych wyrazów (anotowany).

ogon listy – lista powstała po obcięciu jej pierwszego elementu.

Współrzędne morfologiczne – zbiór par postaci {*nazwa kategorii morfologicznej, wartość tej kategorii*}.

Wyraz – ten termin bez dodatkowych określeń rozumiem w sensie intuicyjnym. W zasadzie najczęściej jego znaczenie zbliżone jest do wyrazu **alfabetycznego**, tym niemniej zastrzegam, iż traktuję go mniej ściśle.

Wyraz alfabetyczny – ciąg znaków wyodrębniony w tekście znakami nieliterowymi. Wyraz alfabetyczny jest pojęciem odwołującym się wyłącznie do własności tekstu zapisanych w dostępnej nam reprezentacji. Podział na wyrazy alfabetyczne jest więc jednoznaczny. Wyraz alfabetyczny zachowuje kasztę.

Wyraz morfologiczny – stosuję ten termin w sensie zbliżonym do używanego w pracach [2] i [3]. Jest to jednostka, której przyporządkowujemy osobny zestaw współrzędnych morfologicznych. Pojęcie to odwołuje się do aspektów języka niezawartych w formalnym zapisie, więc podział na wyrazy morfologiczne może zależeć od przyjętej teorii lingwistycznej. Wyrazy morfologiczne są bezkasztove, co reprezentują zapisując je małymi literami.

Przykład:

Mielibyśmy biało-czerwoną flagę.

Wyrazy alfabetyczne: *Mielibyśmy,biało,czerwoną,flagę*.

Wyrazy morfologiczne wg teorii lingwistycznej zastosowanej do anotacji korpusu IPI PAN: *mieli,byśmy,biało,czerwoną,flagę*

Wyrazy morfologiczne wg “szkolnej” teorii lingwistycznej: *milibyśmy, biało, czerwoną, flagę*

Wyrażenie regularne – ciąg znaków (zwany wzorcem), zawierający być może znaki specjalne, określający jednoznacznie zbiór ciągów znaków pasujących do siebie. Istnieją różne składnie wyrażen regularnych, lecz podstawowe znaki specjalne są zazwyczaj takie same:

- kropka oznacza dowolny znak,
- gwiazdka powtórzenie poprzedzającego znaku dowolną liczbę razy,
- plus powtórzenie poprzedzającego znaku co najmniej jeden raz,
- znak zapytania wystąpienie poprzedzającego znaku lub nie.

Najprostsze składnie pozwalają tylko na użycie gwiazdki w znaczeniu dowolnego ciągu znaków.

Przykłady:

$m \cdot a$ – dowolny ciąg znaków rozpoczynający się od znaku m i kończący znakiem a

mu^+ – wyrazy $mu, muu, muuu, \dots$

W pracy przez wyrażenie regularne rozumiem też analogiczną konstrukcję na wyższym poziomie abstrakcji – działającą na wyrazach, a nie na znakach. Będzie to więc ciąg znaków specjalnych i wyrazów jednoznacznie określający zbiór ciągów wyrazów. W tym przypadku nie ma ogólnie przyjętych standardów, jakkolwiek w notacjach z którymi się spotkałem stosowane są znaki specjalne identyczne, jak w przypadku wyrażen regularnych na poziomie znaków.

Oto prosty przykład takiego wzorca:

Domek był [PRZYMIOTNIK]⁺ i [PRZYSŁÓWEK]? [PRZYMIOTNIK]

Pasuje do zdania

Domek był biały i przytulny,

jak też do zdania

Domek był niski drewniany zgrabny i bardzo przytulny.

(dla przejrzystości przykładu pominąłem kwestię interpunkcji).

Część I. Istniejące narzędzia

Rozdział 1

Funkcje dostępne w programach tworzących konkordancje i ich znaczenie

W tym rozdziale omówię funkcje, jakie udostępniają przeanalizowane przeze mnie programy, zwracając uwagę na ich użyteczność, trudność implementacji oraz powiązania z innymi funkcjami i własnościami korpusu. Podzielone zostały na niezależne od zawartości korpusu (implementowalne na każdym korpusie) oraz zależne od niej (implementowalne tylko na korpusach posiadających odpowiednie cechy).

1.1. Niezależne od struktury i zawartości korpusu

1.1.1. Możliwość zadawania zapytań wielowyrazowych

Zdarza się często, że użytkownikowi potrzebne jest nie odnalezienie wystąpienia konkretnego wyrazu, ale całej kilkuwyrazowej frazy, np. **Czerwony Kapturek**. W takim wypadku odnalezienie tylko jednego z wyrazów w oczywisty sposób niewiele daje.

Z technicznego punktu widzenia implementacja wyszukiwania fraz nie jest zbyt skomplikowana. W najbardziej naiwnej, lecz skutecznej, wersji wystarczy odnajdywać wszystkie wystąpienia pierwszego wyrazu i sprawdzać każde z nich pod kątem wystąpienia kolejnych pasujących wyrazów bezpośrednio po nim. W przypadku wyszukiwania tekstu w dokładnie takiej postaci jak zadana w zapytaniu (bez analizy morfologicznej) jest to jeszcze prostsze, ponieważ cały ciąg wraz ze spacjami potraktować można jako jeden wyraz.

Obsługa zapytań wielowyrazowych otwiera drogę do dalszego zwiększania mocy narzędzia, jeśli można stosować wyrażenia regularne na poziomie wyrazów oraz istnieje możliwość wyszukiwania bądź ignorowania znaków interpunkcyjnych.

1.1.2. Możliwość użycia wyrażeń regularnych

Na poziomie znaków

Obsługa wyrażeń regularnych na poziomie znaków oznacza możliwość ich użycia do definiowania poszczególnych wyrazów zapytania. Jest ona praktycznie niezbędna do obsługi tych korpusów języków syntetycznych, które nie mają możliwości odwołania do formy hasłowej ani funkcji automatycznego wyszukiwania wszystkich możliwych form zadanego wyrazu. W większości takich przypadków wystarczy możliwość zignorowania końcówki w szukanym wyrazie

– nakazania wyszukania wszystkich wyrazów zaczynających się zadaniem ciągiem. Tym niemniej pełniejsza obsługa wyrażeń regularnych może być bardzo użyteczna we wszystkich innych korpusach, choćby nawet dla zaoszczędzenia wpisywania znaków przy poszukiwaniu długich a rzadkich wyrazów.

Do tej kategorii funkcji kwalifikuję też technicznie bliźniaczą obsługę wyrażeń regularnych w zadawanych wartościach kategorii morfologicznych, np. `POS=AD.*` pasować będzie zarówno do wyrazów z wartością kategorii *POS* równą *ADJ*, jak i *ADV*.

Pamiętać jednak należy, że dopasowanie wyrażenia regularnego jest operacją znacznie bardziej kosztowną czasowo niż zwykle porównanie ciągów (choć w przypadku wielokrotnego poszukiwania tego samego wzorca koszt ten można zredukować budując automat rozpoznający dane wyrażenie). Oprócz tego dopuszczenie wyrażeń regularnych może skomplikować lub uniemożliwić wykorzystanie indeksów, wspomagających wyszukiwanie w dużym zbiorze danych. To wszystko powoduje, że wyszukiwanie fragmentów pasujących do zapytania zawierającego wyrażenie regularne może trwać znacznie dłużej niż do prostego zapytania. Celowe wydaje się użycie takich algorytmów, które w przypadku zapytań prostych działają tak szybko, jakby wyrażenia regularne nie istniały, ale radzą sobie (być może znacznie wolniej) również z zapytaniami zawierającymi te wyrażenia.

Część programów oferuje tylko namiastkę wyrażeń regularnych, np. tylko znak odpowiadający dowolnemu ciągowi znaków.

Na poziomie wyrazów

Dopuszczenie użycia wyrażeń regularnych na poziomie wyrazów znacznie zwiększa moc narzędzi wyszukiwujących, ale też niesie ze sobą jeszcze większe zagrożenie znacznym przedłużeniem operacji wyszukiwania. Tu również należałoby postąpić tak, jak w przypadku wyrażeń regularnych na poziomie znaków: zadbać, by istnienie możliwości zadawania zapytań zaawansowanych nie zmniejszyło efektywności działania zapytań prostych. Użytkownik powinien też mieć świadomość, że jeśli zbuduje skomplikowane zapytanie zawierające wyrażenia regularne na obu poziomach, to nawet doskonałemu algorytmowi wyszukiwanie może zająć dużo czasu.

Oprócz tego implementacja wyrażeń regularnych na poziomie wyrazów, szczególnie dopasowywania wzorca typu *od n do k wystąpień wyrazu w*, wiąże się z koniecznością podjęcia przez programistę pewnych decyzji determinujących zachowanie algorytmu wyszukiwania w pewnych szczególnych przypadkach.

1.1.3. Możliwość wyszukiwania wyrazów występujących w zadanej odległości od siebie

Do badań użyteczne może być odnajdywanie zadanych wyrazów w kontekście innych (KWIC – Keyword In Context), np. *afery* w bliskim sąsiedztwie *korupcyjna*. Kolejność wyrazów może nie grać roli, powinno dać się zdefiniować dopuszczalną maksymalną odległość od kontekstu. Takie wyszukiwanie trwa nieco dłużej od zwykłego, gdyż za każdym razem po odnalezieniu głównego wyrazu trzeba obejrzeć kilka wyrazów w przód i wstecz.

Funkcja ta da się zastąpić odpowiednio użytymi wyrażeniami regularnymi na poziomie wyrazów, jeśli składnia jest wystarczająco bogata. Można na przykład użyć zapytania postaci:

```
(afery [powtórz-COKOLWIEK-od-0-do-4-razy] korupcyjna)
OR (korupcyjna [powtórz-cokolwiek-od-1-do-4-razy] afery)
```

aby uzyskać *afery* w odległości do 4 wyrazów od *korupcyjna*.

Oczywiście możliwość bardziej intuicyjnego zadania zapytania jest zaletą.

1.1.4. Możliwość sortowania wyników

Część interfejsów do korpusów udostępnia możliwość sortowania otrzymanej konkordancji według zadanych kryteriów. Najczęściej jest to ciąg odnaleziony (pasujący do zapytania) lub któryś wyraz na lewo bądź prawo od niego. W przypadku korpusów zawierających znaki interpunkcyjne na prawach zwykłych wyrazów może to prowadzić do niezbyt użytecznego sortowania po znakach interpunkcyjnych, jeśli wystąpią w małej odległości od odnalezionego fragmentu.

Sortowanie nie jest dużym wyzwaniem technicznym. Przy użyciu odpowiedniego algorytmu (np. Heapsort) można dość efektywnie posortować nawet bardzo duże w stosunku do dostępnej pamięci porcje danych. Wymaga to jednak tymczasowego przechowania wyników wyszukiwania, a więc uniemożliwia wykonanie całej operacji jednorazowo, bez użycia dodatkowej pamięci.

1.1.5. Możliwość wyszukiwania w wynikach

Po otrzymaniu wynikowej konkordancji użytkownik może zapragnąć uściślić swoje zapytanie, ale nie budując je od nowa, a tylko wyszukując w dotychczas otrzymanych wynikach. Żaden ze znanych mi programów nie udostępnia takiej możliwości. Implementacja mogłaby nastęrczać pewnych trudności, zwłaszcza w przypadku programów działających w architekturze *wiele klientów-serwer*, np. interfejsów WWW, gdzie problemem może być konieczność przechowywania danych o sesji pracy danego użytkownika (wyników ostatnio zadanego zapytania). Jeśli zapytanie miało wiele wyników, a serwer obsługuje naraz wielu takich użytkowników, może wtedy zaistnieć problem wielkości przechowywanych danych. Można temu łatwo zapobiec ograniczając liczbę wyświetlanych trafień dla jednego zapytania. Stoi to jednak w sprzeczności z ideą wyszukiwania w kolejnych krokach zawężających liczbę trafień kolejnymi uściśleniami. Znaczna część potencjalnych wyników mogłaby wtedy zostać arbitralnie obciążona już w pierwszym kroku.

1.2. Zależne od struktury i zawartości korpusu

Przechodzimy tu z poziomu badania wyrazu alfabetycznego na poziom badania wyrazu morfologicznego, czyli na wyższy poziom abstrakcji, na którym interesuje nas już nie tylko zewnętrzna reprezentacja w postaci ciągu znaków, ale też cechy gramatyczne badanych jednostek.

Wymaga to istnienia w korpusie opisu morfologicznego każdego wyrazu. Generowanie go podczas wyszukiwania byłoby szczytem nieefektywności – musiałby dla każdego wyrazu być w ciągu “czasu życia” korpusu tworzony tyle razy, ile razy korpus był przeszukiwany. Dane te muszą więc w jakiś sposób zostać dołączone do tekstu korpusu.

Niektóre mniejsze korpusy anotowane są ręcznie, co daje dużo lepszą dokładność, lecz wymaga olbrzymiego nakładu ludzkiej pracy. Większe korpusy anota się więc automatycznie przy pomocy odpowiedniego oprogramowania. Jakość anotacji zależy oczywiście od stopnia skomplikowania danego języka naturalnego i użytych algorytmów; według danych autorów korpusów współczynnik błędów osiąga zazwyczaj kilka procent.

1.2.1. Możliwość odwołania do formy hasłowej

Pierwszym krokiem w kierunku wykorzystania zawartej w korpusie informacji morfologicznej jest możliwość wyszukiwania wyrazów o danej formie hasłowej. Kwestia co jest formą hasłową danego wyrazu morfologicznego może podlegać dyskusji; w większości przypadków jednak dla użytkownika powinno być to intuicyjnie jasne, np. bezokolicznik dla czasownika, mianownik liczby pojedynczej dla rzeczownika itp. Przyjąć oczywiście można dowolne założenia, jeśli tylko użytkownik korpusu jest ich świadom.

Ta funkcja może być używana do odnalezienia wszystkich form gramatycznych danego wyrazu lub ciągu wyrazów, np. *białemu domkowi*, *białym domkiem* dla zapytania typu

HASŁOWA='biały' HASŁOWA='domek'.

Jest to wielkie ułatwienie szczególnie dla języków o bogatej morfologii, takich jak słowiańskie czy ugrofińskie.

Jego brak może być mniej dotkliwy, gdy istnieje możliwość wyszukiwania wyrażeń regularnych na poziomie wyrazów: można wtedy na przykład zażądać wyrazów rozpoczynających się od *biał*, by odnaleźć wszystkie formy pochodne od *biały*. Widać jednak niedoskonałość tej metody: niepotrzebnie odnaleziony zostanie też *białawy*. Problem sprawia też *domek* z racji ruchomego 'e', z którym wprowadzić też można sobie poradzić za pomocą odpowiednio skonstruowanego wyrażenia regularnego (jeśli wystarczająco silne wyrażenia są zaimplementowane), lecz jest to już bardziej skomplikowane i może zniechęcić mniej doświadczonego użytkownika.

1.2.2. Możliwość odwołania do współrzędnych morfologicznych

Kolejnym krokiem w kierunku większej użyteczności jest możliwość użycia w zapytaniu współrzędnych morfologicznych poszukiwanego wyrazu. Technicznie nie jest to wiele bardziej skomplikowane w realizacji niż obsługa formy hasłowej, która może być tak naprawdę traktowana na równi z innymi współrzędnymi morfologicznymi.

Problem sprowadza się więc (poza obecnością odpowiednich danych w korpusie) do wprowadzenia obsługi wielu współrzędnych morfologicznych zamiast jednej. Wymaga to ponazywania tych współrzędnych, co komplikuje język zapytań. W korpusach z opisem pozycyjnym dodaje się tylko jedną współrzedną, będącą długim ciągiem zawierającym w sobie w określonym formacie cały opis morfologiczny. Aby odwołać się do konkretnych kategorii należy odfiltrować z niego odpowiednie pozycje wyrażeniem regularnym.

Możliwość pytania o wartości poszczególnych kategorii morfologicznych pozwala na pełniejsze wykorzystanie zgromadzonej w korpusie wiedzy. Szczególnie użyteczne jest badanie części mowy w językach analitycznych, co pozwala np. na odnalezienie wystąpień angielskiego wyrazu *can* w funkcji rzeczownikowej (*puszka*), a nie czasownikowej (*móc*).

1.2.3. Możliwość odwołania do znaków interpunkcyjnych

Ta funkcja ma sens tylko w przypadku zapytań wielowyrazowych, ponieważ wyszukiwanie samych znaków interpunkcyjnych nie wydaje się być specjalnie przydatne. Użytkownik może jednak mieć potrzebę rozróżnienia frazy typu

mały, domek
od
mały domek

(zauważmy że ta pierwsza może być częścią zdania *Ogródek był mały, domek jeszcze mniejszy*, które nie satysfakcjonuje nas w przypadku wyszukiwania frazy *mały domek*).

Z drugiej strony, wygodne może być dopasowanie do zapytania
miły przytulny

zarówno fragmentów, w których przymiotniki są rozdzielone przecinkiem, jak i średnikiem albo niczym – choćby dla wygody zadawania zapytania.

Widać więc, że zarówno całkowite ignorowanie znaków interpunkcyjnych, jak też traktowanie ich na równi ze zwykłymi wyrazami ma swoje wady. W pierwszym przypadku nie da się właściwie rozpoznać zdania z przykładu pierwszego, w drugim wyszukiwarka może nie odnaleźć wszystkich intuicyjnie pasujących wystąpień.

Wydaje się więc celowe przechowywanie informacji o znakach interpunkcyjnych tak samo jak o wyrazach, ale oddanie użytkownikowi kontroli nad sposobem ich traktowania.

Za obsługę odwołań do znaków interpunkcyjnych uznałem w porównaniu możliwość odnalezienia znaków interpunkcyjnych i odróżnienia ich od innych wyrazów, przez na przykład opatrzenie ich odpowiednim parametrem. Ponieważ moje wnioski na ten temat wynikały najczęściej wyłącznie z testów praktycznych, mogą być w niektórych przypadkach błędne.

1.2.4. Możliwość odwołania do metadanych

Większość korpusów zawiera informacje o źródle, z którego pochodzi odnaleziony fragment tekstu. Może to być tylko nazwa podkorpusu, ale też autor, tytuł, data wydania, lub też dane o typie tekstu (proza, poezja, publicystyka itp.). W przypadku korpusów języka mówionego mogą to być też informacje typu wiek, płeć albo wykształcenie osoby mówiącej.

Możliwość odwołania się do metadanych pozwala zawęzić wyszukiwanie do interesującej badacza grupy tekstów, jak również porównywać pewne cechy tekstów z różnych grup.

Technicznie metadane mogą być traktowane w składni zapytań na równi ze współrzędnymi morfologicznymi – wtedy domyślnie wyszukiwane są wszystkie wystąpienia, a w celu zawężenia poszukiwań należy określić dodatkowy parametr wyszukiwanego wyrazu. Tu też zdarza się zapis wszystkich metadanych w postaci jednego pozycyjnego ciągu.

Możliwe jest też inne podejście – w Węgierskim Korpusie Narodowym oraz w FIDA wybiera się, na których podkorpusach zostanie wykonane całe zapytanie. Ma to więcej sensu jeśli metadanych jest mało; w przypadku wielu ich kategorii mogłoby być to nieczytelne i niewygodne.

1.2.5. Możliwość wykorzystania uzgadniających się zmiennych

W korpusach posiadających interfejs umożliwiający odwoływanie się do współrzędnych morfologicznych (a w ogólności do dowolnych współrzędnych opisywanych wyrazów) oraz możliwość zadawania zapytań wielowyrazowych można rozważyć dodanie obsługi zmiennych w zapytaniach.

Najprostszym przykładem użyteczności tego udoskonalenia może być badanie gramatycznego związku zgody. Załóżmy, że poszukujemy wszystkich par *przymiotnik - rzeczownik* pozostających w tym związku. Jego istotą jest bycie w tym samym przypadku. Dla języka polskiego należałoby więc zapytanie sformułować tak:

```
([PRZYMIOTNIK,Mian] [RZECZOWNIK,Mian]) OR ([PRZYMIOTNIK,Dop] [RZECZOWNIK,Dop])  
OR ([PRZYMIOTNIK,Cel] [RZECZOWNIK,Cel]) OR ([PRZYMIOTNIK,B] [RZECZOWNIK,B]) OR  
([PRZYMIOTNIK,Narz] [RZECZOWNIK,Narz]) OR ([PRZYMIOTNIK,Msc] [RZECZOWNIK,Msc]),
```

czyli wypisać alternatywę wszystkich możliwych przypadków. Jest to pracochłonne i nieczytelne nawet przy tak prostym zapytaniu. Gdyby użytkownik zapragnął umożliwić występowanie innych wyrazów pomiędzy szukanym rzeczownikiem a przymiotnikiem, zapytanie stałoby się nieakceptowalnie długie i niezrozumiałe.

Wprowadzenie zmiennych umożliwia intuicyjne zadanie tego zapytania – niejako użycie w nim sformułowania *taki sam*:

[PRZYMIOTNIK, $x1$] [RZECZOWNIK, $x1$]

Nie zostało określone, jakiemu przypadkowi odpowiada zmienna $x1$. Jednak ta sama zmienna występuje w definicjach obu poszukiwanych wyrazów – a więc przypadek ma być dowolny, ale ten sam dla obydwu wyrazów.

W przypadku bardziej skomplikowanych zapytań zmienne mogą być jeszcze bardziej pożyteczne. Następczą też jednak trudności implementacyjnych – konieczne staje się utrzymywanie tablicy przypisań zmiennych oraz jej obsługa w przypadku nawrotów z błędnych ścieżek poszukiwania, co może być uciążliwe, zwłaszcza w podejściu nierekurencyjnym.

W żadnym z analizowanych przeze mnie programów nie spotkałem się z możliwością użycia zmiennych.

Rozdział 2

Przegląd programów tworzących konkordancje

W niniejszym rozdziale przedstawiam krótkie opisy programów tworzących konkordancje, do jakich udało mi się dotrzeć – ich funkcje, dostępność oraz, w przypadku narzędzi powiązanych z konkretnym korpusem, krótką charakterystykę tegoż korpusu.

2.1. IMS Corpus Workbench i CQP

Jest to opracowany w *Institut für Maschinelle Sprachverarbeitung* na uniwersytecie w Stuttgarcie zestaw narzędzi do pełnotekstowego wyszukiwania w korpusie. Strona projektu znajduje się pod adresem <http://www.ims.uni-stuttgart.de/projekte/CorpusWorkbench>. Sercem zestawu jest *Corpus Query Processor (CQP)* – interpreter języka zapytań. Korpus obsługiwany przez CQP musi być zapisany w specjalnym formacie. Binaria CQP do zastosowań badawczych dostępne są za darmo pod warunkiem podpisania umowy licencyjnej, zabraniającej m.in. zastosowań komercyjnych, redystrybucji oraz dekompilacji.

Cechy języka zapytań według jego autorów:

- brak ograniczenia na liczbę atrybutów elementu korpusu
- obsługa wyrażeń regularnych na poziomie wartości atrybutów elementów korpusu
- obsługa wyrażeń regularnych na poziomie ciągu elementów korpusu
- częściowa obsługa anotacji strukturalnej (np. w SGML) korpusu
- “incremental concordancing” (interpretuję to jako możliwość wyszukiwania w wynikach, lecz nie odnalazłem w dokumentacji wyjaśnienia)
- możliwość zastosowania zapytania do wszystkich elementów listy
- ‘wirtualne atrybuty’, czyli możliwość uruchamiania zewnętrznych aplikacji w czasie analizy zapytania
- obsługa zapytań na tekstach równoległych w różnych językach

Możliwości modułu prezentacji wyników:

- definiowalny rozmiar prezentowanego kontekstu wyszukanych elementów
- różnorodne metody sortowania linii wynikowej konkordancji
- wyliczanie częstości występowania, np. kombinacji słów
- konkordancje równoległe (wielojęzyczne)
- obsługa formatu html i latex
- zapamiętywanie historii zapytań

CQP działa w trybie linii komend, pod kontrolą systemu Linux lub Solaris. Dlatego też mniej obeznani z komputerem użytkownicy wykorzystują go najczęściej za pośrednictwem interfejsów – stron WWW lub osobnych programów generujących właściwą treść zapytania oraz kontaktujących się z centralnym serwerem, jak na przykład w przypadku programu GCQP, służącego do obsługi Czeskiego Korpusu Narodowego.

Powyższe informacje pochodzą wyłącznie z dokumentacji. Nie podejmowałem wysiłku podpisywania formalnej umowy wyłącznie po to, by przetestować narzędzie, zwłaszcza że większość funkcji mogłem sprawdzić pośrednio, testując oparte na CQP programy, np. opisany niżej interfejs WWW do *The Oslo Corpus of Bosnian Texts*.

2.2. Czeski Korpus Narodowy – interfejs dostępny przez WWW

Pod adresem <http://ucnk.ff.cuni.cz/verejny.html> możemy wykonać prostą konkordancję na podkorpusie PUBLIC (dostępnej bez ograniczeń części Czeskiego Korpusu Narodowego, zawierającej ok. 20 mln wyrazów). Konkordancja zawiera wyłącznie wyrazy w dokładnie takiej postaci, jak podana w zapytaniu – nie jest dokonywana żadna analiza morfologiczna. Długość kontekstu można wybrać (maksymalnie 60 znaków). Definiując wyszukiwany wyraz można używać wyrażeń regularnych. Zapytanie nie może się składać z wielu wyrazów. Wypisywane jest do 20 pierwszych trafień lub opcjonalnie do 20 losowych trafień. Druga opcja oznacza, że wyniki kolejnych wywołań dla tego samego zapytania mogą być różne. Nie ma możliwości sortowania wyników.

Powyższe informacje pochodzą z badań własnych.

2.3. Czeski Korpus Narodowy – program łączący się z serwerem Korpusu za pomocą protokołu Telnet

Użytkownik korzysta ze specjalnego programu GCQP z interfejsem graficznym, pozwalającego zadawać zapytania dotyczące podkorpusu SYN2000 (współczesny język czeski; podkorpus zawiera ok. 100 mln wyrazów). Program łączy się z serwerem za pomocą protokołu `telnet`. Samo oprogramowanie klienckie dostępne jest bez ograniczeń. Aby otrzymać login i hasło do serwera, należy się zarejestrować oraz złożyć (zwykłą pocztą) oświadczenie o przyjęciu do wiadomości zasad ograniczonego korzystania z uzyskanych danych, m.in. tylko do zastosowań niekomercyjnych.

Definiując wyszukiwany wyraz można używać wyrażeń regularnych. Można wyszukiwać wyrazy o zadanej formie hasłowej lub – w pewnym zakresie – o żądanych współrzędnych morfologicznych. Informacja morfologiczna w Korpusie jest zakodowana jednym ciągiem znaków o stałej długości. Znając jego kodowanie można skonstruować wyrażenie regularne filtrujące konkretne formy lub zbiory form. Na przykład:

[tag='...4.*'] filtruje wszystkie bierniki (piąta pozycja koduje przypadek), a [tag='A...1.*'] wszystkie przymiotniki w mianowniku.

Można zadawać zapytania wielowyrazowe, ale w przeciwieństwie do CQP nie ma możliwości użycia wyrażeń regularnych na poziomie wyrazów. Takie zubożenie interfejsu w stosunku do oprogramowania bazowego jest pewnym zaskoczeniem. Istnieje jednak intuicyjna obsługa wynajdowania wyrazów w zadanej odległości od siebie, mogąca częściowo wyrównywać ten brak.

Wyniki można sortować po odnalezionym ciągu, według lewego bądź prawego kontekstu (w tym – według np. trzeciego wyrazu na lewo od odnalezionego), według formy hasłowej lub ciągu tagów morfologicznych. Można też oraz uzyskać informację na temat źródła każdego odnalezionego wystąpienia. Nie ma możliwości dalszego wyszukiwania w wynikach. Gotową konkordancję można wyeksportować.

Powyższe informacje pochodzą z dokumentacji [9].

2.4. Węgierski Korpus Narodowy – ogólnodostępny interfejs WWW

Pod adresem <http://corpus.nytud.hu/mnsz/> udostępniono interfejs do wyszukiwania w całości Węgierskiego Korpusu Narodowego, zawierającego ponad 153 miliony wyrazów, podzielonego na następujące podkorpusy: *Prasa, Literatura, Nauka, Oficjalne i Osobiste*. Interfejs oparty jest na CQP. Aby otrzymać login i hasło należy potwierdzić, że będzie się używać Korpusu tylko do celów badawczych, edukacyjnych bądź prywatnych, oraz zobowiązać się do podania nazwy i adresu projektu w wykorzystujących go pracach. W przeciwieństwie do korpusu czeskiego, zobowiązania nie trzeba wysyłać zwykłą pocztą – wszystko można zrobić za pośrednictwem strony WWW.

Wyszukiwanie definiuje się specyfikując pojedynczy wyraz alfabetyczny lub parę wyrazów, która ma znaleźć się w korpusie w promieniu do 5 wyrazów od siebie. Można też nakazać traktowanie wpisanego w zapytaniu tekstu jako rdzenia – wówczas dopasowane zostaną wszystkie formy pochodne tego rdzenia (w języku węgierskim oznacza to niemal wyłącznie dodawanie końcówek). Możliwe jest też zdefiniowanie wymagań co do części mowy i formy gramatycznej poszukiwanego wyrazu. Można to uczynić dwójako: szybciej za pomocą kodów lub bardziej intuicyjnie przez wybranie poszczególnych wymagań z list dostępnych wartości.

Definiując wyszukiwany tekst można używać wyrażeń regularnych, choć nie jest to wspomniane w żadnym dostępnym na stronie opisie. Podobnie tylko analizując wyświetlane ponad konkordancją właściwe zapytania do CQP (generowane na podstawie wyboru opcji z list) można wydedukować, jak w bardziej zaawansowany sposób (np. ze spójnikami logicznymi) pytać o części mowy i formy gramatyczne.

Interesującym udogodnieniem jest możliwość dowolnego wyboru podkorpusów, które zostaną przeszukane. Wypisywane może być do 500 losowo wybranych trafień. Wyniki można sortować według wyrazu przed odnalezionym lub po odnalezionym. Długość kontekstu może być regulowana – od 1 do 10 wyrazów.

Powyższe informacje pochodzą z umieszczonej na stronie WWW dokumentacji oraz badań własnych.

2.5. Słowacki Korpus Narodowy – ogólnodostępny interfejs WWW

Pod adresem <http://korpus.juls.savba.sk> znajduje się interfejs dostępu do Słowackiego Korpusu Narodowego. Aby z niego skorzystać, należy potwierdzić (wybierając odpowiedni

odnośnik na stronie), że otrzymane dane wykorzystana się tylko do celów naukowych, badawczych i edukacyjnych, a nie komercyjnych, jak również zobowiązać się do podawania korpusu jako źródła oraz niepodejmowania prób zdobycia nieautoryzowanego dostępu do systemu komputerowego.

Interfejs sprawia wrażenie opartego na CQP, lecz nie udało mi się odnaleźć na stronie żadnej informacji na ten temat. Również nie ma informacji na temat składni zapytań, więc wszelkie moje wnioski pochodzą z prób praktycznych. Dodatkowym utrudnieniem jest wyraźna niestabilność i błędy w działaniu interfejsu: często zamiast konkordancji ukazuje się pusta strona, a czasem odnajdywany jest wyraz niezwiązany z zapytaniem. Z racji tego oraz zupełnego braku opisu, programu tego nie uwzględniam w ogóle w podsumowaniu.

Można używać wyrażeń regularnych na poziomie znaków. Użycie zapytań wielowyrazowych powoduje dziwne zachowania programu, co nie pozwala wyciągnąć jasnych wniosków na temat ich obsługi.

Dla każdego odnalezionego wystąpienia można wyświetlić dane o źródle oraz szerszy kontekst. Nie ma możliwości sortowania ani wyszukiwania w wynikach.

Jak wspominałem, powyższe informacje pochodzą wyłącznie z badań własnych.

2.6. Korpus języka słoweńskiego Instytutu Frana Ramovša

Pod adresem http://bos.zrc-sazu.si/a_beseda.html znajduje się interfejs WWW do przeszukiwania opracowywanego w Instytucie Frana Ramovša korpusu tekstów słoweńskich, zawierającego aktualnie 48 milionów wyrazów.

Interfejs nie został oparty na CQP, ponieważ serwer Instytutu działa na systemie Windows NT. Zamiast tego opracowany został własny, znacznie uboższy interfejs NEVA, bazujący na wcześniej tam napisanym na edytorze EVA.

Jedynym dopuszczalnym znakiem specjalnym jest gwiazdka, oznaczająca dowolny ciąg znaków, której jednak nie można użyć bez kontekstu co najmniej kilku znaków zdefiniowanych jednoznacznie (zasady nie są dla mnie jasne). Można zadawać zapytania wielowyrazowe, ale nie można używać wyrażeń regularnych na poziomie wyrazów.

Nie odnalazłem żadnej informacji o możliwości odwołania do formy hasłowej bądź współrzędnych morfologicznych. Da się zdefiniować na których podkorpusach zapytanie ma być wykonane.

Nie ma możliwości sortowania konkordancji wynikowej. Można uzyskać informację o pochodzeniu każdego odnalezionego wystąpienia oraz szerszy kontekst. Nie można wyszukiwać w wynikach.

Powyższe informacje pochodzą z umieszczonej na stronie WWW dokumentacji oraz badań własnych.

2.7. Korpus języka słoweńskiego FIDA

Korpus FIDA jest udostępniany komercyjnie. Za darmo można jedynie testować interfejs, na który jest wtedy nakładane ograniczenie liczby wyświetlanych rezultatów do 10; pozostałe funkcje są w pełni dostępne. Interfejs znajduje się pod adresem <http://www.fida.net>.

Nie ma możliwości zadawania zapytań wielowyrazowych – spacja działa jak spójnik logiczny “i”. Można używać znaku zapytania i gwiazdki w znaczeniu dowolnego znaku/ciągu znaków. Podobnie jak w Czeskim Korpusie Narodowym przy dostępie telnetowym, można wyszukiwać wyrazy o zadanej formie hasłowej lub o zadanej postaci dość skomplikowanego pozycyjnego ciągu opisującego formę gramatyczną. Na przykład `#2p??z?i*` oznacza polecenie

wyszukania wszystkich przymiotników żeńskich w mianowniku, o pozostałych współrzędnych dowolnych. Dozwolone jest użycie spójników logicznych oraz spójników oznaczających znajdowanie się dwóch wyrazów w tekście w określonej (niewielkiej) odległości od siebie.

Przy wyszukiwaniu rozszerzonym można zdefiniować, które podkorpusy mają być przeszukane.

Powyższe informacje pochodzą z umieszczonej na stronie WWW dokumentacji oraz badań własnych.

2.8. Chorwacki Korpus Narodowy – ogólnodostępny interfejs WWW

Pod adresem <http://www.hnk.ffzg.hr/30me.htm> wykonać można konkordancję na ciągle budowanym “30–milionowym korpusie współczesnego języka chorwackiego”, który w tej chwili zawiera ok. 9 milionów elementów.

Można wyszukiwać wyłącznie pojedyncze wyrazy. Nie jest dokonywana żadna analiza morfologiczna. Jedyńm obsługiwany znak specjalny jest %, oznaczający dowolny ciąg znaków.

Nie ma możliwości dalszego działania na wynikach. Można jedynie uzyskać informację o kodzie źródła i szerszy niż w samej konkordancji kontekst (250 znaków).

Powyższe informacje pochodzą z umieszczonej na stronie WWW dokumentacji oraz badań własnych.

2.9. The Oslo Corpus of Bosnian Texts – interfejs WWW

Pod adresem <http://www.tekstlab.uio.no/Bosnian/Corpus.html> można wykonać konkordancję na korpusie tekstów bośniackich. Przedtem należy się zarejestrować (przez Internet) i złożyć oświadczenie, że korpusu będzie się używać wyłącznie do celów akademickich i niekomercyjnych, nie zdradzi się nikomu przydzielonego hasła oraz poda się nazwę i adres korpusu w wykorzystujących go pracach.

Interfejs oparty został na CQP i dziedziczy wszystkie jego możliwości składniowe w zakresie obsługi wyrażeń regularnych na obu poziomach. Nie odnalazłem informacji o możliwości odwołania do formy hasłowej.

W strukturze korpusu każdy wyraz niesie ze sobą metadane – kod źródła. Można więc zapytać o wszystkie wystąpienia zadanego wyrazu w konkretnym tekście lub, filtrując ów kod wyrażeniami regularnymi, w wybranym roku.

Maksymalna długość kontekstu jest ograniczona do 500 znaków. Wyników nie da się sortować ani w nich wyszukiwać. Można jedynie przejrzeć skromne dane o źródle. Dla zapytań odnajdujących wiele różnych form (np. z różnymi końcówkami) zamiast konkordancji można wyświetlić dystrybucję tych form.

Powyższe informacje pochodzą z umieszczonej na stronie WWW dokumentacji oraz badań własnych.

2.10. Korpus PWN – próbka dostępna przez WWW

Pod adresem <http://korpus.pwn.pl> możemy wykonać konkordancję na fragmencie korpusu ogólnego PWN. Udostępniony fragment zawiera ok 1,8 mln wyrazów, w tym ok. 70 tysięcy różnych.

Niezależnie od formy wyrazu w zapytaniu, wyszukiwane są wszystkie formy gramatyczne (bez eliminacji homonimii). W zapytaniu zdefiniować można tylko jeden wyraz – w przypadku wpisania wielu wyszukane zostaną one osobno, a wyniki zsumowane. Te zachowania można zmienić ujmując treść zapytania w cudzysłów – wtedy wyszukiwane są wystąpienia ciągu sąsiednich wyrazów w zadanej kolejności i w dokładnie takich formach gramatycznych. Nie ma pełnej obsługi wyrażen regularnych – można używać tylko gwiazdki (zastępuje dowolny ciąg znaków) i znaku zapytania (dowolny znak). Nie można ich stosować wewnątrz cudzysłowów. Użycie znaków specjalnych wyłącza analizę morfologiczną tak samo, jak ujęcie tekstu w cudzysłów.

Nie ma możliwości definiowania żądanych współrzędnych morfologicznych poszukiwanego wyrazu.

Wypisywane może być do 200 trafień. Według opisu *wyszukiwarka przygotowuje konkordancje cytatów wybranych losowo z różnych źródeł*, lecz wbrew temu kolejne wywołania dla tego samego zapytania dają identyczne wyniki. Wyniki można sortować według pierwszego wyrazu lewego lub prawego kontekstu, oraz według znalezionej sekwencji. Można uzyskać krótką informację na temat źródła każdego z odnalezionych wystąpień. Długość kontekstu może być regulowana – od 0 do 99 wyrazów. Opis mówi wprawdzie o ograniczeniu do 30 wyrazów, lecz praktyka temu przeczy.

Powyższe informacje pochodzą z umieszczonej na stronie WWW dokumentacji oraz badań własnych.

2.11. Korpus PWN – próbka dostępna na płycie CD

Na płycie CD dostępna jest inna próbka korpusu PWN. Zawiera ona prawie 4 miliony wyrazów. Dołączony do korpusu serwer konkordancji (wyłącznie wersja dla systemu Windows) pozwala na zadawanie zapytań przez przeglądarkę WWW działającą na komputerze użytkownika. Interfejs jest nieco inny niż w przypadku próbki na stronie WWW, również funkcjonalnie.

Wylimitowano niedogodność osobnego wyszukiwania wyrazów w zapytaniu nie ujętym w cudzysłowy. Dzięki temu można odnaleźć wszystkie formy gramatyczne całej frazy: zapytanie w mózg odnajdzie w *mózgach*, w *mózgu*, w *mózg*, w *mózgi*, a nie tak jak w wersji WWW – wszystkie wystąpienia w oraz wszystkie wystąpienia jakiejś formy rzeczownika *mózg*. Nie ma obsługi wyrażen regularnych – można jedynie używać gwiazdki i znaku zapytania, ale tylko w zapytaniach jednowyrazowych (choć opis o tym nie wspomina). W razie potrzeby możliwa jest wyłączenie analizy morfologicznej (wyszukiwania wszystkich form gramatycznych).

Tu również nie jest eliminowana homonimia ani nie ma możliwości definiowania żądanych współrzędnych morfologicznych poszukiwanego wyrazu. Istnieje natomiast możliwość wyszukiwania danego wyrazu w bliskim kontekście innego.

Wypisywane może być do 500 trafień. Wyniki można sortować według ostatniego wyrazu lewego kontekstu, pierwszego wyrazu prawego kontekstu oraz według znalezionej sekwencji. Można uzyskać krótką informację na temat źródła każdego z odnalezionych wystąpień. Długość kontekstu może być regulowana – od 1 do 10 wyrazów.

Powyższe informacje pochodzą z zawartej na płycie dokumentacji oraz badań własnych.

2.12. Korpus IPI PAN – wersja dostępna na CD

Na płycie CD udostępniona została wstępna wersja korpusu IPI PAN, wraz z graficzną i tekstową wersją programu do przeszukiwania korpusów *Poliqarp*. Ta wersja korpusu zawiera

ponad 70 milionów segmentów, co odpowiada ponad 364 tysiącom różnych form hasłowych.

Język zapytań jest wzorowany na CQP. Obsługuje zapytania wielowyrazowe oraz rozbudowane wyrażenia regularne na obu poziomach. Możliwe jest odwołanie do formy hasłowej oraz współrzędnych morfologicznych, jak również do szerokiego zestawu metadanych. Oprócz tego dostępnych jest wiele technicznych udogodnień, takich jak możliwość nakazania ignorowania kaszty.

Istnieje możliwość sortowania wyników, wyświetlenia szerszego kontekstu poszczególnych trafień, oraz informacji o źródle. Można też zażądać, by w dopasowaniu i kontekście zamiast reprezentacji zewnętrznej odnalezionych wyrazów wyświetlane były ich formy hasłowe oraz znaczniki morfologiczne.

Powyższe informacje pochodzą z dokumentacji [5].

2.13. Korpus IPI PAN – próbka dostępna przez WWW

Pod adresem <http://korpus.pl> umieszczono próbkę efektów pracy IPI PAN – tzw. korpus **sample**, zawierający ponad 15 milionów segmentów, co przekłada się na nieco mniejszą liczbę wyrazów alfabetycznych oraz na ok. 217 tysięcy różnych form hasłowych.

Wersja WWW działa wprawdzie na ograniczonym fragmencie korpusu, ale obsługuje ten sam rozbudowany język zapytań, co opisana wcześniej wersja dostępna na płycie CD.

Bogate są możliwości sortowania wyników – rosnąco lub malejąco, a fronte lub a tergo. Przy sortowaniu brane są pod uwagę nie słowa w znaczeniu intuicyjnym, ale wyrazy alfabetyczne, którymi w strukturze tego korpusu są również znaki interpunkcyjn. W szczególności więc sortując a tergo według lewego kontekstu na początku otrzymamy te trafienia, w których lewy kontekst kończy się znakami przestankowymi, lub np. nawiasami.

Powyższe informacje pochodzą z dokumentacji [5] oraz z testów własnych.

2.14. British National Corpus – próbka dostępna przez WWW

Pod adresem <http://sara.natcorp.ox.ac.uk/lookup.html> można wykonać konkordancję na całości British National Corpus. BNC zawiera 100 milionów wyrazów.

Zapytanie może być wielowyrazowe, zawierać wyrażenia regularne na poziomie wyrazów oraz specyfikację części mowy, w której funkcji ma wystąpić wyszukiwane słowo. Wyrażenia regularne na poziomie wyrazów nie są wprawdzie obsługiwane, ale istnieje specjalny znak `_` oznaczający dowolne słowo.

Sztucznym ograniczeniem w wersji ogólnodostępnej jest liczba wyświetlanych trafień (50, wybranych losowo, lecz w krótkim odstępie niezmiennych) oraz długość kontekstu. Nie ma żadnej możliwości dalszej pracy z wynikami oprócz sprawdzenia źródła.

Powyższe informacje pochodzą z zamieszczonej na stronie WWW dokumentacji oraz z testów własnych.

Rozdział 3

Podsumowanie możliwości analizowanych programów

3.1. Przegląd

W tabelce znajduje się zbiorcze podsumowanie możliwości analizowanych programów przeszukujących korpusy. Kolejne kolumny oznaczają:

1. obsługę zapytań wielowrazowych
2. obsługę wyrażeń regularnych na poziomie znaków
3. obsługę wyrażeń regularnych na poziomie wyrazów
4. możliwość sortowania wyników
5. możliwość szukania w wynikach
6. możliwość odwołania do formy hasłowej
7. możliwość odwołania do współrzędnych morfologicznych
8. możliwość wyszukiwania bądź ignorowania znaków interpunkcyjnych
9. możliwość odwołania do metadanych
10. możliwość użycia zmiennych
11. możliwość odnajdywania zadanych wyrazów w kontekście innych,

natomiast kolejne wiersze:

1. IMS Corpus Workbench i CQP
2. Czeski Korpus Narodowy – interfejs dostępny przez WWW
3. Czeski Korpus Narodowy – program łączący się z serwerem Korpusu
4. Węgierski Korpus Narodowy
5. Korpus języka słoweńskiego Instytutu Frana Ramovša
6. Korpus języka słoweńskiego FIDA

7. Chorwacki Korpus Narodowy – ogólnodostępny interfejs WWW
8. Korpus PWN – próbka dostępną przez WWW
9. Korpus PWN – próbka dostępną na płycie CD
10. Korpus IPI PAN – próbka dostępną przez WWW
11. Korpus IPI PAN – wersja dostępną na CD
12. British National Corpus – próbkę dostępną przez WWW
13. The Oslo Corpus of Bosnian Texts – interfejs WWW.

Trzeba mieć świadomość, że dostępność funkcji zależnych od zawartości korpusu (w sensie podziału dokonanego w rozdziale 1) jest siłą rzeczy wymuszona przez cechy korpusów, na których programy działają. Dlatego w wierszu dotyczącym niezwiązanego z żadnym konkretnym korpusem CQP występują puste kratki przy funkcjach, które na odpowiednio bogatym korpusie mogą być przy użyciu tego narzędzia implementowane. Znak '+/-' oznacza, że funkcja jest dostępna w ograniczonym zakresie, np. tylko bardzo proste wyrażenia regularne, lub automatyczne dopasowywanie wszystkich form zadanego wyrazu zamiast możliwości zdefiniowania formy hasłowej.

	Wielowy- razowe	Reg. znak	Reg. wyraz	Sort.	Szuk. w wyn.	Forma hasłowa	Współrz. morf.	Inter- punkcja	Meta- dane	Zmienne	KWIC
CQP	✓	✓	✓	✓						✗	✓
ČNK WWW	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
ČNK Telnet	✓	✓	✗	✓	✗	✓	✓	✗	✗	✗	✓
MNSZ	+/-	✓	✗	✓	✗	✓	✓	✓	✓	✗	✓
Ramovš	✓	+/-	✗	✗	✗	✗	✗	✗	✓	✗	✗
Fida	✗	+/-	✗	✗	✗	✓	✓	✗	✓	✗	✓
HNK	✗	+/-	✗	✗	✗	✗	✗	✗	✗	✗	✗
PWN WWW	+/-	+/-	✗	✓	✗	+/-	✗	✗	✗	✗	✗
PWN CD	✓	+/-	✗	✓	✗	+/-	✗	✗	✗	✗	✓
IPI PAN WWW	✓	✓	✓	✓	✗	✓	✓	✗	✓	✗	✓
IPI PAN CD	✓	✓	✓	✓	✗	✓	✓	✗	✓	✗	✓
BNC	✓	✓	✗	✗	✗	✗	+/-	✗	✗	✗	✗
OCBT	✓	✓	✓	✗	✗	✗	✓	✓	✓	✗	✗

3.2. Wnioski

Po dokonaniu przeglądu kilkunastu interfejsów do korzystania z korpusów stwierdzam, iż wiele z nich nie ma dobrej morfologicznej obsługi. Wprawdzie wszystkie pozwalają na użycie pewnej formy (nieraz bardzo ubogiej) wyrażeń regularnych na poziomie znaków i większość pozwala zadawać zapytania wielowyrazowe, ale już obsługa wyrażeń regularnych na poziomie wyrazów jest rzadkością. Trzeba przyznać, że spora część programów zastępuje ten brak jakąś formą obsługi odnajdywania wyrazów w kontekście innych wyrazów (Keywords In Context), tym niemniej jest to rozwiązanie ograniczające możliwości narzędzia. Z drugiej jednak strony, możliwe jest, że użytkownicy tych programów wolą używać prostych i intuicyjnych pojęć takich jak *w niewielkiej odległości od*, niż konstruować silne, ale skomplikowane wyrażenia regularne.

Zaskakuje też znaczny udział korpusów i programów nie pozwalających na odwołania do formy hasłowej i innych współrzędnych morfologicznych. To może być prostym skutkiem braku odpowiednich danych w korpusie; tym niemniej jest to znacząca niedogodność przy badaniach, szczególnie brak dostępności formy hasłowej. Oczywiście przy użyciu wyrażeń regularnych na poziomie wyrazów można próbować radzić sobie z tym problemem, np. wyszukując wszystkie wyrazy o zadanym temacie i dowolnej końcówce; jednak nie zawsze da się osiągnąć oczekiwany efekt, na przykład w językach słowiańskich, gdzie występują oboczności tematyczne bądź zjawisko zupełnego niepodobieństwa tematów w różnych formach tego samego wyrazu.

Żaden z programów nie oferuje możliwości wykorzystania zmiennych w budowanym zapytaniu. To intrygujące, bowiem po implementacji tej funkcji w systemie Pescador stwierdzam, że nie okazała się nastroczać specjalnych trudności.

Żaden nie daje też możliwości wyszukiwania w wynikach. To jest bardziej zrozumiałe, ponieważ konieczne w takim przypadku przechowywanie wyników zadawanych zapytań może być poważnym obciążeniem serwera konkordancji. Również implementacja nie musi być prosta, jeśli program nie został zaprojektowany od początku tak, by tę funkcję obsługiwać.

Większość interfejsów narzuca użytkownikom ograniczenia w korzystaniu ze zgromadzonych w korpusie danych. Zawężenie prezentowanego kontekstu jest spowodowane ochroną praw autorskich do zgromadzonych tekstów. Ograniczenie liczby odnajdowanych trafień jest raczej dążeniem do ochrony serwerów przed zbyt dużym obciążeniem.

Część II. Pescador – przykładowy system

Rozdział 4

Założenia wstępne

- System działać ma na systemie Linux. Możliwe jest dołączenie go do którejś z pozwalających na to istniejących dystrybucji na bootowalnych płytach CD, aby dało się z niego korzystać bez konieczności posiadania systemu Linux na swoim komputerze.
- System ma być dostępny na licencji GPL.
- Dokumentacja systemu (niniejsza praca) ma być dostępna na licencji FDL (Free Documentation License).
- Główny moduł systemu ma być w miarę możliwości uniwersalny językowo, tj. pozwalać na obsługę wielu języków. Tylko moduły zewnętrzne (obcy – analizator morfologiczny i własny – konwerter wyników analizatora) mogą zależeć od badanego języka.
- System ma charakter przykładowy; nie ma ambicji być oprogramowaniem dla końcowego użytkownika. Z tego względu nie ma potrzeby skupiać się np. na interfejsie ani wyrafinowanej obsłudze błędów. Nie jest też konieczne implementowanie wszystkich popularnych funkcji innych programów tego typu, a skoncentrować się należy na tym, czego w innych programach brakuje.

Rozdział 5

Wybór języka i narzędzi programistycznych

5.1. Język programowania

Do zaprogramowania głównej części systemu wybrałem funkcyjny język SML. Wybór ten może wydawać się zaskakujący – z przetwarzaniem języka naturalnego zazwyczaj kojarzy się język Prolog. Jednak mechanizm dopasowywania wzorca dostępny w SML w zupełności wystarcza do wyszukiwania w reprezentującej dane wejściowe strukturze, mimo że nie jest dostępny prologowy mechanizm uzgadniania termów. W przeciwieństwie do Prologu, w SML bardzo łatwo definiuje się i wykorzystuje skomplikowane i uniwersalne struktury danych. Oprócz tego, niemalą częścią programu właściwego są parser pliku wejściowego oraz parser zapytania, które zdecydowanie łatwiej jest napisać w języku funkcyjnym.

Drugą ewentualnością był język nieco niższego poziomu, na przykład C/C++; nie było jednak potrzeby takiej uniwersalności narzędzia, a byłoby wtedy znacznie więcej pracy z dopracowaniem technicznych szczegółów implementacji, którymi SML zajmuje się automatycznie (definiowanie struktur danych, obsługa pamięci). Pewną przewagą tego rozwiązania byłoby łatwiejsze debugowanie programu, w języku funkcyjnym rzeczywiście czasem dość uciążliwe.

Wybór akurat SML, a nie Haskell'a podyktowany był głównie przyzwyczajeniem oraz dobrymi doświadczeniami z tym pierwszym przy poprzednim projekcie (uczącym się programie konwersacyjnym porozumiewającym się w języku esperanto).

5.2. Kompilator

Rozpoczynając poszukiwania kompilatora SML starałem się odnaleźć taki, który umie skompilować program również pod system Windows. Brałem pod uwagę 3 kompilatory:

- ML Kit, działający wyłącznie na systemie Linux i kompilujący tylko na niego.
- MLton, posiadający wersje na systemy Linux oraz Cygwin (emulator Linuksa dla Windows). Pozwalałoby to wygenerować kompilat w postaci windowsowego pliku .exe, który jedynie wymagałby do działania biblioteki cygwin.dll.
- New Jersey SML, działający na obu systemach i potrafiący na nie kompilować.

Ostatecznie zrezygnowałem z warunku istnienia wersji dla Windows i wybór padł na kompilator MLton. Kompiluje on kod źródłowy całościowo, co pozytywnie wpływa na efektywność

działania programu wynikowego. Według testów przeprowadzonych przez autorów na kilkudziesięciu różnych programach źródłowych, MLton jest w większości przypadków od kilkudziesięciu procent do kilkudziesięciu razy szybszy od porównywanej konkurencji.

Rozdział 6

Schemat ogólny systemu

W znajdującym się na następnej stronie schemacie kolor niebieski oznacza dane i operacje zewnętrzne w stosunku do Pescadora, czarny – wewnętrzne.

Główny moduł systemu (4) na wejściu otrzymuje:

- treść korpusu w formacie czystego tekstu (opisanym w 7.1), którego odpowiednie fragmenty będą pojawiały się w wyjściowej konkordancji w dokładnie takiej postaci (pod względem interpunkcji itp.), w jakiej były w oryginale
- wynik analizy morfologicznej (anotację) w formacie opisanym w 7.2; w tych danych będzie się odbywało wyszukiwanie

Dane w obu plikach są powiązane – każda anotowana jednostka w wyniku analizy posiada dowiązanie do swojej pozycji w tekście korpusu.

W przypadku analizy korpusu anotowanego konieczny jest moduł (2), konwertujący zawartość korpusu na czysty tekst. Moduł taki powinien być dość prosty, a przede wszystkim można się spodziewać, że taka aplikacja będzie częścią oprogramowania korpusu. Jako taka nie jest przez mnie uznawana za część Pescadora.

Niezbędny jest też moduł (3), konwertujący tekst korpusu na format wewnętrzny Pescadora i ten już jest częścią systemu – inną dla każdego języka i korpusu.

W przypadku analizy korpusu anotowanego również konieczny jest moduł (2), w tym przypadku zapewne jeszcze prostszy, a w szczególnych przypadkach zbędny (gdy korpus jest zapisany w formacie pliku tekstowego). Aby otrzymać tekst anotowany należy przetworzyć korpus analizatorem morfologicznym (1) (niebędącym oczywiście częścią Pescadora) oraz otrzymany wynik modułem konwersji wyników (3) przygotowanym dla tego analizatora.

Mając już przygotowane dane wejściowe, można zadawać zapytania. Moduł główny (4) odbiera je, przetwarza i generuje wyniki – konkordancje. Kolejne zapytanie można zadać na pierwotnych danych wejściowych (wczytanych z pliku), lub na wynikach poprzedniego zapytania.

Rozdział 7

Formaty danych wejściowych

7.1. Format pliku zawierającego tekst korpusu

1. Zawiera wyłącznie znaczący tekst (nie ma żadnych formatowań)

Pierwotnie za wartą rozważenia uznałem koncepcję przyjęcia jako format wejściowy CES. Jest to format SGML-owy zgodny z zaleceniami TEI, opracowany przez grupę EAGLES we współpracy z projektem MULTTEXT. Stosowany był dotąd w kilku projektach korpusowych, między innymi w Multext-East. Zgodnie z nim (a raczej z jego XML-ową wersją XCES – więcej na ten temat w pracy [1]) jest oznaczony korpus języka polskiego IPI PAN.

Założyłem jednak, że praktycznie każdy istniejący format korpusu da się bez większego wysiłku przekonwertować na czysty tekst. Przyjęcie jakichkolwiek konwencji formatowania pliku wejściowego zmuszałoby do przygotowywania modułu konwersji tekstu korpusu dla każdego wykorzystywanego korpusu zapisanego niezgodnie z tymi konwencjami. Użycie formatu tekstowego pozwala założyć, że konwersją zajmie się oprogramowanie korpusu, które powinno zawierać odpowiednie narzędzia do jego obsługi, na przykład arkusze XSLT dla korpusów zapisanych w XML.

Dlatego też zrezygnowałem z koncepcji dostosowywania oprogramowania do formatu CES/XCES. Wykorzystanie informacji zawartej w korpusie innej niż jego właściwa treść (tekst) byłoby w programie niewielkie, a nakład pracy konieczny do implementacji wczytywania korpusu zapisanego w CES/XCES – duży.

Anotację strukturalną korpusu można częściowo zachować operując znakami końca linii (jedyne dopuszczalne rodzaje formatu dla pliku wejściowego). Można więc zachować podział tekstu na paragrafy, ale np. podział na rozdziały czy zwrotki zaginie.

2. Nie zawiera wyrazów przeniesionych do nowego wiersza

Dopuszczenie w tekście wyrazów przenoszonych do nowego wiersza mogłoby prowadzić do powstania niejednoznaczności, na przykład bez dodatkowej wiedzy słownikowej nie da się odróżnić następujących sytuacji:

czworo-
nożny

polsko-
niemiecki

Oba przypadki będą traktowane jako dwa osobne wyrazy alfabetyczne oddzielone myślnikiem.

3. Zapisany jest w (dowolnym) kodzie jednobajtowym

Ograniczenie to wynika z technicznych możliwości zastosowanych narzędzi programistycznych, a w szczególności użytego gotowego modułu obsługi wyrażeń regularnych. Poprawienie lub napisanie od nowa tego modułu jest ważnym polem do dalszego rozwoju systemu w kierunku większej uniwersalności językowej.

7.2. Format wewnętrzny anotacji

Przez format wewnętrzny rozumiem sposób, w jaki dane reprezentowane są w plikach roboczych i w pamięci programu. Nie jest to oczywiście w obu przypadkach dokładnie ta sama reprezentacja: w pamięci dane przechowywane są w SML-owej strukturze danych, a w plikach roboczych w formacie będącym podzbiorem XML-a.

Zaprojektowanie właściwych wewnętrznych formatów kodowania informacji jest bardzo istotnym zagadnieniem z dwóch powodów:

- Program powinien umożliwiać podłączenie modułu analizy morfologicznej (1) i modułu konwersji informacji morfologicznej (3) dla dowolnego języka (z dokładnością do problemów natury notacyjnej). Oczywiście nie uda się tego zrobić idealnie i może się okazać, że pewne cechy struktury jakiegoś języka, które nie zostały przewidziane, nie pozwolą go prawidłowo opisać. Podejmując decyzje wybierałem rozwiązania gwarantujące dobre zachowanie wobec języka polskiego i pokrewnych, zwracając jednak uwagę na potencjalną uniwersalność. Nie są ograniczeniem istniejące w języku kategorie morfologiczne, Pescador bowiem nie robi żadnych założeń co do ich nazw, ilości i dopuszczalnych wartości (poza istnieniem formy hasłowej).
- Wybranie uniwersalnego formatu wewnętrznego może znacząco ułatwić stworzenie modułów konwersji informacji morfologicznej (3).

Zastosowałem koncepcję anotacji zewnętrznej. Takie rozwiązanie nie narusza pliku wejściowego oraz, co dużo ważniejsze, pozwala na wielokrotne anotowanie tego samego tekstu różnymi analizatorami – powstaje wtedy kilka plików z anotacją odnoszących się do jednego pliku wejściowego.

Pliki robocze (w formacie wewnętrznym) tworzone są przez moduł konwersji tagów (3) na podstawie wejściowych plików tekstowych, opisanych w 7.1, oraz informacji morfologicznej, pochodzącej bądź to z korpusu, bądź (w przypadku korpusów bez anotacji) z analizatora morfologicznego. Wymagane jest stworzenie pliku roboczego dla każdego pliku wejściowego. Plik roboczy zawiera całą treść pliku wejściowego, gubiąc tylko szczegóły interpunkcyjne takie jak wielokrotne spacje itp. Same właściwe znaki interpunkcyjne są zachowywane, jest nawet możliwe ich wyszukiwanie w korpusie (np.: “znajdź wszystkie wystąpienia jakiegoś wyrazu przed przecinkiem”).

Można się zastanawiać, czy w takim przypadku przechowywanie obok pliku roboczego również pliku wejściowego ma sens. Jednak z racji dość dużego rozmiaru plików roboczych nie jest to już duży narzut, a pozwala uniknąć komplikacji przy wyświetlaniu kontekstu, związanych z opisaną poniżej niejednoznacznością segmentacji. Ze względu na nią wyszukiwanie poprzedniego wyrazu w pliku roboczym nie jest trywialne, w przeciwieństwie do wyszukiwania w oryginalnym pliku tekstowym.

7.2.1. Problem niejednoznaczności segmentacji

Nie można zakładać, że tekst wejściowy uda się jednoznacznie podzielić na wyrazy morfologiczne. Podaję przykład za Wolińskim [8]:

Miałem miał

Możliwe są tu trzy interpretacje, dające dwie różne segmentacje:

miał(rzeczownik)-em(aglutynacyjna forma czasownika BYĆ) miał(czasownik) – tak jak “Kotam miał”

miał(pseudoimiesłów)-em(aglutynacyjna forma czasownika BYĆ) miał(rzeczownik) – tak jak “Miałem kota”

miałem(rzeczownik w narzędniku) miał(pseudoimiesłów) – tak jak “Piaskiem miał (posypany trawnik)”.

Oczywiście według bardziej klasycznej teorii drugą wersję można również podzielić na dwa wyrazy morfologiczne:

miałem(czasownik w czasie przeszłym) miał(rzeczownik)

Skoro nawet w obrębie jednej metodologii opisu morfologicznego możliwa jest wielość interpretacji, należy w formacie pliku z anotacją zawrzeć możliwość opisania tego.

Dodatkowo fakt, że granica wyrazów morfologicznych może wypadać w środku wyrazu alfabetycznego (więcej przykładów znajduje się w pracy [6]), zmusza do przyjęcia niskopoziomowego systemu adresowania danych w pliku wejściowym. Po dokonaniu wyszukiwania w pliku roboczym występuje potrzeba odwołania się do pliku wejściowego – w celu wypisania w konkordancji kontekstu odnalezionego ciągu. Tak więc informacje zawarte w pliku z anotacją powinny umożliwić szybkie odnalezienie odpowiedniego miejsca w pliku wejściowym. Gdyby adresować numerami kolejnych wyrazów morfologicznych, to odnalezienie takie wymagałoby przeczytania pliku wejściowego od początku i przeanalizowania pod kątem segmentacji. Takie rozwiązanie jest nie do przyjęcia, gdyż po pierwsze jest bardzo nieefektywne (a zapytanie może dać bardzo wiele wyników), a po drugie adresowanie byłoby niejednoznaczne z powodu wspomnianej możliwości istnienia wielu segmentacji.

Dlatego też każdy wyraz opisany morfologicznie w pliku z anotacją zna swój początkowy bajt w pliku wejściowym. Umożliwia to szybkie wyszukanie, jak również poradzenie sobie z problemem niejednoznaczności segmentacji. Opisy wyrazów dla różnych segmentacji będą umieszczane w pliku po kolei. Komplikuje to wyszukiwanie poprzedniego i następnego wyrazu w pliku.

Przykład (oczywiście każdy wyraz niebędący znakiem interpunkcyjnym ma jeszcze opis morfologiczny):

Miałem miał.

012345678901

```
s(0,'miał'),  
s(4,'em'),  
s(0,'miałem'),  
s(7,'miał'),  
s(11,'.')
```

Wyrazy pierwszy oraz drugi pokrywają się z trzecim. Wyraz następny po drugim to nie trzeci, lecz czwarty; stwierdzenie tego nie jest trudne gdy znamy pozycje w pliku oraz długości wyrazów.

7.2.2. Reprezentacja w pamięci podczas pracy programu

Przyjęta reprezentacja wewnętrzna ma postać drzewa o stałej głębokości równej 4.

Korzeniem drzewa jest węzeł typu `Tfile`, mający atrybut typu `string` (nazwa pliku). Węzeł `Tfile` może mieć dowolną liczbę synów typu `Tline` (linii w pliku).

Węzeł typu `Tline` ma atrybut typu `int` (numer linii w pliku) i może mieć dowolną liczbę synów typu `Tword` (wyrazów występujących w tej linii).

Węzeł typu `Tword` może reprezentować:

- jeden wyraz morfologiczny. Ma wtedy następujące atrybuty: liczony w bajtach adres początku wyrazu morfologicznego w pliku wejściowym (typu `int`); tekst wyrazu morfologicznego (`string`); tekst wyrazu alfabetycznego, w którym dany wyraz morfologiczny wystąpił (`string`). Ostatni atrybut jest opcjonalny, jego brak oznacza że dany wyraz morfologiczny stanowi całość alfabetycznego, tzn. drugi atrybut byłby równy pierwszemu. Taki węzeł typu `Tword` może mieć dowolną liczbę synów typu `Tform`.
- jeden znak interpunkcyjny. Wtedy posiada tylko dwa atrybuty: liczony w bajtach adres w pliku wejściowym (typu `int`) oraz jednoelementowy ciąg znaków zawierający ten znak. Taki węzeł nie ma potomków.
- podział linii. Jest to sztuczny symbol wstawiany pomiędzy kolejnymi liniami, uniemożliwiający dopasowanie do wzorca ciągów rozdzielonych pomiędzy dwie linie (chyba, że użytkownik zażąda wystąpienia tego znaku, zawierając we wzorcu element `NL`). Taki węzeł nie ma potomków.

Węzeł typu `Tform` reprezentuje możliwą interpretację morfologiczną wyrazu. Jego atrybuty to forma hasłowa (`string`) oraz współrzędne morfologiczne (lista par typu (`string, string`)).

7.2.3. Reprezentacja w plikach roboczych

Stosowany w plikach roboczy format powyższej struktury w pierwotnym założeniu miał mieć charakter czysto wewnętrzny. Wybór podzbioru XML powoduje jednak, że może być on wygodnie i łatwo czytany przez człowieka i przetwarzany narzędziami stworzonymi dla XML. Otwiera to szerokie możliwości wykorzystania plików roboczych Pescadora.

Stosowany podzbiór XML jest zgodny z poniższym pseudo-DTD:

```
<!ELEMENT file (line+)>
<!ELEMENT line ((ano|punct)+)>
<!ELEMENT ano (int+)>
<!ELEMENT punct EMPTY>
<!ATTLIST file fname CDATA #REQUIRED>
<!ATTLIST line ln CDATA #REQUIRED>
<!ATTLIST ano start CDATA #REQUIRED>
<!ATTLIST ano word CDATA #REQUIRED>
<!ATTLIST ano textword CDATA #IMPLIED>
<!ATTLIST punct start CDATA #REQUIRED>
<!ATTLIST punct word CDATA #REQUIRED>
<!ATTLIST int ???>
```

Nie da się go opisać w pełni formalnym DTD z dwóch względów:

- atrybuty elementu **int** mogą być dowolne – reprezentują one listę dowolnych współrzędnych morfologicznych opisywanego wyrazu morfologicznego. Dowolność ta jest przenoszona na zapytania: jeśli dany wyraz morfologiczny posiada kategorię **kat1** i jej wartość to *war1*, to zostanie odnaleziony zapytaniem nakazującym, by **kat1** przyjęła wartość *war1*. Nie zostanie dopasowany, jeżeli w ogóle nie posiada takiej kategorii, lub przyjmuje ona inną wartość.
- sugerowałyby to pełną obsługę XML, co nie jest prawdą. Pescador nie wczyta bowiem dowolnego pliku w formacie XML zgodnego z odpowiednim DTD. Na przykład element **ano**, nawet jeśli jest pusty, nie może mieć postaci zakończonej `/>`; element **punct** zaś odwrotnie: nie może być zamykany ciągiem `</punct>`. Podyktowane to zostało dążeniem do uproszczenia leksera, wobec braku potrzeby obsługiwanego pełnego formatu XML. O ile bowiem potrzeba przetwarzania plików roboczych Pescadora innymi programami (XML-owymi) może zaistnieć, to potrzeba generowania ich – nie, ponieważ są one tworzone wyłącznie przez dedykowany i będący częścią systemu moduł konwersji anotacji (3).

Rozdział 8

Język zapytań

Kluczową kwestią decydującą o przydatności systemu w pracach lingwistycznych jest opracowanie odpowiedniego języka zapytań. Język ten powinien dawać jak największe możliwości. W trakcie pracy zrezygnowałem z wymogu intuicyjności wychodząc z założenia, że dla użytkowników nie posiadających wiedzy informatycznej możliwe jest stworzenie, wzorem na przykład Węgierskiego Korpusu Narodowego, prostego interfejsu pozwalającego generować zapytania, jak też jednocześnie wpisywać je bezpośrednio.

W systemie Pescador zapytanie może być traktowane jako ciąg oddzielonych spacjami termów, które po kolei dopasowywane są do przeszukiwanego tekstu (rozumianego jako ciąg wyrazów morfologicznych opatrzonych opisami). Każdy z tych termów może zostać dopasowany do dowolnej liczby wyrazów z tekstu (również do zera). Pewnym ukłonem w stronę wygody kosztem formalności zapisu jest rezygnacja z żądania opatrywania nazwą predykatu wzorców definiujących jeden wyraz, opisanych poniżej. Dzięki temu zamiast na przykład WYRAZ(zielone[mu|j]) można napisać po prostu zielone[mu|j].

8.1. Definicja jednego wyrazu

Na tym poziomie stosuje się notację podobną do używanej w CQP. Można zdefiniować:

- Żądany kształt wyrazu alfabetycznego bezpośrednio. Zostanie on potraktowany jako wzorzec w sensie wyrażenia regularnego według składni AWK. Wzorzec zachowuje kaszkę.

```
kalafior
zielone[mu|j]
prac.*
```

- Zestaw żądanych współrzędnych morfologicznych. Poszczególne pary *kategoria-wartość* zapisywane są ze znakiem = lub ~ i są od siebie oddzielone spacjami. Całość jest ujęta w nawiasy kwadratowe. Aby traktować drugi element pary jako wyrażenie regularne według składni AWK, należy użyć znaku ~ ; chcąc traktować go literalnie, należy użyć = . Przykłady:

[pos='adj' case='N'] – kategoria 'pos' ma przyjąć wartość 'adj', a 'case' – 'N'

[pos~'ad'] – kategoria 'pos' ma pasować do wyrażenia regularnego 'ad' (tj. być ciągiem zawierającym podciąg 'ad')

[lemma~'wi.a'] – kategoria 'lemma' ma pasować do wyrażenia regularnego 'wi.a' (np. *witać, dźwigać, wikary*)

[lemma='wi.a'] – kategoria 'lemma' ma przyjąć dokładnie wartość 'wi.a' (co zapewne raczej się nie zdarzy)

Dzięki istnieniu sztucznej kategorii `word` (przyjmującej wartość równą treści wyrazu morfologicznego) żądany kształt wyrazu można zdefiniować dwojako. Zapytania `malinowy` oraz `[word~'malinowy']` oznaczają bowiem dokładnie to samo. Warto zauważyć różnicę w stosunku do zapytania `[word='malinowy']`. W tym ostatnim przypadku wzorzec będzie traktowany dosłownie, a nie sensie wyrażenia regularnego, a więc na przykład nie zostanie dopasowany do wyrazu *malinowym*.

8.2. Predykaty specjalne

- koniunkcja dwóch wyrazów: `AND(w1,w2)`
- alternatywa dwóch wyrazów: `OR(w1,w2)`
- powtórzenie wyrazu *w* od *n* do *k* razy: `REP(w,n,k)`
- dowolny wyraz (nie będący znakiem interpunkcyjnym): `ANY`
- zezwolenie i żądanie przejścia w środku wzorca do nowej linii: `NL`

Predykaty specjalne mogą zostać dopasowane do ciągów wyrazów morfologicznych o dowolnej długości. Mogą być zagnieżdżane, tj. ich argumentami wyrazowymi mogą być inne predykaty. Nie mogą jednak nimi być wyrazy zadane bezpośrednio – zamiast `OR(bo,ponieważ)` należy napisać `OR([word='bo'], [word='ponieważ'])`.

8.3. Gramatyka języka zapytań

```
zapytanie      := slowo | slowo zapytanie
slowo          := regex | slowo_og
slowo_og       := ANY | REP(slowo_og,integer,integer) | OR(slowo_og,slowo_og) |
                AND(slowo_og,slowo_og) | NL | [ ciag_warunkow ] | []
ciag_warunkow := warunek | warunek ciag_warunkow
warunek        := atrybut='wartosc' | atrybut~'regex' | atrybut = #cyfra
cyfra          := 0|1|2|3|4|5|6|7|8|9|0
```

gdzie `regex` jest wyrażeniem regularnym według składni AWK, a `wartosc` dowolnym ciągiem znaków do wyszukania.

8.4. Obsługa zmiennych

Obsługa zmiennych jest funkcją nowatorską, nie występującą w żadnym znanym mi oprogramowaniu do obsługi korpusu. Zmiennych jest 10, oznaczone są symbolami #0, #1 do #9. Występować mogą jako wartości atrybutów (kategorii morfologicznych). Mogą zostać dopasowane do dowolnej wartości w odnalezionym ciągu wyrazów, lecz zawsze ta sama zmienna musi odpowiadać tej samej wartości. W ten sposób można na przykład stworzyć zapytanie wyszukujące wyrazy o dowolnym, ale tym samym dla obydwu przypadku. Przykład z opisem znajduje się w 1.2.5.

8.5. Traktowanie znaków interpunkcyjnych

Kierując się rozważaniami z 1.2.3, w systemie Pescador zastosowałem pseudointeligentne podejście do znaków interpunkcyjnych. Polega ono na tym, że znaki interpunkcyjne w przeszukiwanym tekście są ignorowane, chyba że wystąpią w zapytaniu. W związku z tym w tekście

Mało było, ale nie zabrakło: zostały trzy.

zarówno zapytanie

było ale,

jak i

było , ale

zostaną dopasowane do fragmentu *było, ale*.

Zapytanie

zabrakło ANY

zostanie natomiast dopasowane do fragmentu *zabrakło: zostały*.

Oczywiście wszystko to ma miejsce tylko przy założeniu, że w anotacji badanego korpusu zachowane jest rozróżnienie między znakami interpunkcyjnymi a pozostałymi wyrazami.

8.6. Traktowanie podziałów wierszy

Fragmenty tekstu rozdzielone pomiędzy dwa wiersze pliku wejściowego nie są dopasowywane do zapytań, chyba że użytkownik tego jawnie zażąda umieszczając w zapytaniu predykat NL. W tekście

... wjechał do garażu.

Następnego ranka ...

nie zostanie więc odnaleziony wzorzec

garażu Następnego,

ale zostanie odnaleziony

garażu NL Następnego.

To wszystko przy założeniu, że w anotacji korpusu badanego zachowany jest podział na wiersze, tj. wyrazy *garażu* i *Następnego* znajdują się w różnych elementach `line` pliku roboczego.

8.7. Przykłady zapytań

Przykładowe zapytania mają opisaną semantykę tylko przy założeniu istnienia w korpusie odpowiednich kategorii morfologicznych oraz ich wartości.

- `REP('klo.*', 1, 1000)` – od 1 do 1000 wyrazów pasujących do wyrażenia regularnego `klo.*`
- `REP([pos='adj' case='N|A'], 2, 3)` – 2 lub 3 przymiotniki w mianowniku lub bierniku
- `REP(OR([pos='adj' case='N'], [pos='adj' case='A']), 2, 3)` – jak wyżej
- `REP(AND([pos='adj'], OR([case='N'], [case='A'])), 2, 3)` – jak wyżej
- `ANY` – dowolny wyraz (ignoruje znaki interpunkcyjne)
- `OR(REP([pos='verb'], 2, 2), REP(ANY, 6, 7))` – albo dwa czasowniki, albo 6 lub 7 dowolnych wyrazów

- [pos='adj' case=#1] [pos='noun' case=#1] – para przymiotnik-rzeczownik w tym samym przypadku
- [pos='adj' case=#1] [pos='noun' case=#2] – odpowiada zapytaniu [pos='adj'] [pos='noun'], gdyż z racji różności zmiennych oznacza dowolność wartości kategorii case

Rozdział 9

Formaty wyjściowe i eksport

Z powodów wspomnianych w rozdziale 4, nie przywiązywałem wielkiej wagi do formatów wyjściowych. Przykładowy interfejs działa w trybie tekstowym, wypisując w kolejnych wierszach znalezione fragmenty tekstu z 30-znakowym kontekstem. Kontekst od znalezionej fragmentu oddzielony jest znakiem '|'. Wszystkie znaki końca linii występujące w tekście zamieniane są na spacje. W ostatnim wierszu podana jest liczba odnalezionych fragmentów. Oto przykład: (fragmenty tekstu pochodzą z pliku wygenerowanego przeze mnie do celów testowych na podstawie internetowego archiwum *Tygodnika Powszechnego*; ze względów edytorskich prawe konteksty zostały sztucznie skrócone)

```
znamiona ich majątku? Czy są |to ci, którzy - jak| na początku lat 80. - w  
omiast, sorry, liturgia, Msza |Święta ma swoje prawa|. Jeżeli ktoś wchodzi z  
kiedy tu przyjeżdżała, czuła |to samo: irracjonalny, podskórny strach|, który  
edy tu przyjeżdżała, czuła to |samo: irracjonalny, podskórny strach|, który  
4 units found.
```

Dodatkowo istnieje możliwość wyeksportowania wyników (za pomocą komendy `p`). Format pliku wyjściowego identyczny jest z formatem wewnętrznym, opisanym w 7.2.3. Dzięki temu plik ten może zostać ponownie wczytany do dalszych badań za pomocą komendy `'1'`. Każdy z odnalezionych segmentów reprezentowany jest przez jedną linię logiczną w pliku wyjściowym. Kontekst nie jest eksportowany. Znaki interpunkcyjne nie wymienione w zapytaniach (ignorowane podczas pierwotnego wyszukiwania) nie są eksportowane. Plik wyjściowy odwołuje się do tego samego pliku wejściowego i tych samych w nim adresów, co plik, na którym zadawano zapytanie, którego wyniki zostały wyeksportowane.

Rozdział 10

Dokumentacja użytkowa

10.1. Uruchomienie serwera

Serwer jest uruchamiany za pomocą polecenia

```
./pesstart.
```

Należy je wydać po zmianie katalogu na ten, w którym umieszczone zostały pliki systemu. Serwer wyświetli komunikat dopiero po uruchomieniu klienta (interfejsu), który zacznie się z nim komunikować.

10.2. Obsługa interfejsu

Interfejsem użytkownika jest skrypt perlowy, uruchamiany przez wydanie polecenia

```
./pesif.pl
```

w katalogu, w którym umieszczone zostały pliki systemu. Po uruchomieniu ukazuje się (jeśli działa też serwer) zachęta systemowa:

```
:
```

Widząc znak zachęty, można wydawać polecenia. Polecenie składa się z dwóch części: jednego znaku (komendy) oraz ciągu znaków (parametru). Komenda może być oddzielona od parametru dowolną liczbą, w tym zerem, spacji. Niektóre komendy nie wymagają parametru i ignorują go, jeśli się jednak pojawi.

Dostępne komendy:

- l *nazwa* – wczytuje do pamięci plik z anotacją o nazwie *nazwa*. Na tym pliku będą się odbywały odtąd operacje wyszukiwania, aż do momentu wczytania nowego pliku.
- q *zapyt* – wyszukuje w pliku fragmenty pasujące do zapytania *zapyt*.
- r *zapyt* – wyszukuje w wynikach ostatniego zapytania te, które pasują do *zapyt*.
- s – wyświetla informację o aktualnie wczytanym pliku.
- x – zatrzymuje serwer oraz interfejs. Do dalszego działania konieczne będzie ponowne uruchomienie obu programów.
- e – zatrzymuje interfejs bez zatrzymywania serwera. Po ponownym uruchomieniu interfejsu będzie on działał tak, jak gdyby go nie wyłączano; w szczególności nie będzie konieczności ponownego wczytywania pliku z anotacją.

- p – eksportuje wyniki ostatniego zapytania do pliku pescador.exp, nadpisując jego poprzednią zawartość.
- h – wyświetla informację o dostępnych komendach, podobną do niniejszej.

Rozdział 11

Konwertery wejściowe

11.1. Brak anotacji

Zastosowana struktura systemu pozwala niewielkim kosztem umożliwić korzystanie z niego również na korpusach nieanotowanych morfologicznie. Do tego celu służy moduł generujący plik w formacie zgodnym z 7.2 wyłącznie na podstawie czystego tekstu. Po wykonaniu skryptu należy w pierwszej linii pliku wynikowego w elemencie `file` zmienić na właściwą nazwę oryginalnego pliku z tekstem atrybut `fname`. Wygenerowany plik nie zawiera żadnej informacji morfologicznej ani form hasłowych, ale w tym ograniczonym zakresie da się w nim nadal wyszukiwać.

11.2. SAM

Konwerter do plików wyjściowych z programu SAM dr Krzysztofa Szafrana pobiera dane wejściowe z pliku `wyniki.sam`, tworzonoego przez SAMa pracującego w trybie wsadowym. Dane wyjściowe kierowane są na standardowe wyjście. Po wykonaniu skryptu należy w pierwszej linii pliku wynikowego w elemencie `file` zmienić na właściwą nazwę oryginalnego pliku z tekstem atrybut `fname`.

Informacje z SAMa nie są konwertowane dokładnie. Przyjąłem następujące uproszczenia i udoskonalenia:

- rodzaje *m1*, *m2*, *m3* rozróżniane przez SAMa zostały połączone w jeden rodzaj *m*
- w liczbie mnogiej rodzaje *m1* i *-m1* zostały zastąpione bardziej intuicyjnymi i znanymi nazwami *mos* i *nmos*
- ignorowane są oznaczenia ' oraz ", przewidziane "dla form zróżnicowanych w wypadkach wyjątkowych, gdy odpowiednie formy oznaczone liczbami bez tych znaków mają dla przeważającej liczby leksemów te same wykładniki" (za [7])
- ignorowane są interpretacje deprecjatywne dla mianownika liczby mnogiej, czyli np. przymiotnik *złośliwe* nie będzie interpretowany jako rodzaj męski (jak w wyrażeniu *złośliwe chłopcy*)
- odrzucane są informacje o grupach deklinacyjnych i koniugacyjnych
- rozbudowane jest rozpoznawanie form leksemu *on*. W [7], oraz skutkiem tego w SAMie, nie są one rozróżniane, a czytelnik jest odsyłany po szczegóły do tabelki we wstępie

do książki. Konwerter rozpoznaje i przydziela właściwe kategorie gramatyczne dla każdej formy

Wszelkie znaki nie będące częścią wyrazów alfabetycznych są kwalifikowane jako znaki przestankowe (opatrywane znacznikiem `punct`), przy czym są traktowane pojedynczo, tzn. wielokropek wystąpi trzykrotnie jako kropka. Cały tekst jest traktowany jako jedna linia w sensie formatu wewnętrznego Pescadora.

Rozdział 12

Dokumentacja techniczna

12.1. Struktura

12.1.1. Serwer

Serwer jest programem wykonywalnym, napisanym w języku SML, skompilowanym kompilatorem MLton, wersja 20040227. Strukturalnie jego kod źródłowy podzielony jest na następujące moduły:

- **datatypes** – zawiera definicje struktur danych, stosowanych w programie, między innymi **Tfile**, reprezentującej plik wejściowy w pamięci, oraz **Tword**, reprezentującej jeden wyraz morfologiczny wraz z ewentualną anotacją. Ich szerszy opis znajduje się w 7.2.2.
- **regexp** – definiuje dwie funkcje:
 - **compile** kompilującą wyrażenie regularne podane w postaci ciągu znaków
 - **matches** sprawdzającą, czy ciąg znaków pasuje do skompilowanego wyrażenia regularnego

Jest to jedyny moduł w systemie niebędący mojego autorstwa. Dystrybuowany jest razem z kompilatorem New Jersey SML. Udało się go bez większych problemów skompilować MLtonem.

- **typutils** – udostępnia kilka prostych funkcji do operowania na strukturach danych zdefiniowanych w module **datatypes**, takich jak wyłuskiwanie poszczególnych elementów struktury oraz zliczanie podelementów.
- **print** – udostępnia bardzo proste funkcje do wypisywania ciągów oraz liczb; głównym sensem ich istnienia jest zaoszczędzenie pisania w kodzie źródłowym. Właściwe funkcje SML-a służące do wypisywania tekstu mają bowiem skomplikowane nazwy.
- **file** – definiuje następujące funkcje, docelowo operujące na strumieniu wejściowym, wykorzystywane w module **lex**:
 - **streamLen** podającą długość strumienia
 - **endOfStream** sprawdzającą, czy strumień został przeczytany do końca
 - **input1** wczytującą jeden znak z początku strumienia
 - **lookahead** sprawdzającą znak na początku strumienia bez pobierania go stamtąd

- **lex** – definiuje funkcję `lexer`, zamieniającą ciąg znaków (pobierany przez inne moduły z pliku wejściowego w formacie opisanym w 7.2) na ciąg symboli typu `Ttoken`, gotowy do interpretacji przez parser.
- **pars** – definiuje funkcję `pars`, parsującą ciąg symboli z wyjścia leksera i zwracającą strukturę typu `Tfile`.
- **querypars** – definiuje funkcję `parse`, zamieniającą ciąg znaków reprezentujący zapytanie na listę elementów typu `Tworddef` – czyli na postać zrozumiałą dla modułu wyszukującego `finder`.
- **finder** – definiuje funkcję `find`, wyszukującą w podanej strukturze typu `Tfile` segmenty pasujące do zapytania podanego jako lista elementów typu `Tworddef`. Funkcja ta zwraca listę list elementów typu `Tword`. Każda lista reprezentuje jedno odnalezione wystąpienie ciągu pasującego do wzorca.
- **export** – definiuje funkcję `export`, dostającą listę list elementów typu `Tword` oraz nazwę pliku z oryginalnym tekstem korpusu, i na tej podstawie generującą plik `pescador.exp` w formacie roboczym, opisanym w 7.2.3. Taki plik może zostać zachowany i ponownie wczytany później. Może służyć do przechowywania lub przygotowania do dalszego przetwarzania wyników zapytania.
- **stable** – definiuje następujące funkcje, operujące na tablicy symboli reprezentującej przypisane podczas wyszukiwania zmienne:
 - `varKnown` sprawdzającą, czy zmienna o podanym numerze jest zdefiniowana w tablicy symboli
 - `varSet` ustawiającą nową wartość zmiennej o podanym numerze
 - `intersect` wyznaczającą przecięcie dwóch traktowanych jako zbiory list ciągów znaków
- **shower** – definiuje funkcję `show`, pobierającą listę list elementów typu `Tword` oraz nazwę pliku z oryginalnym tekstem korpusu, i na tej podstawie wypisującą wyniki wyszukiwania na ekranie.
- **main** – odpowiedzialny za wykonanie głównej pętli programu, wczytującej komendy z kolejki FIFO `pescador.in`, interpretującej je i wykonującej.

Szerszy opis parametrów i działania poszczególnych funkcji znajduje się w komentarzach w kodzie programu.

12.1.2. Klient – przykładowy interfejs tekstowy

Interfejs kliencki `pesif.pl` jest bardzo prostym skryptem napisanym w języku Perl. W pętli pobiera on ze standardowego wejścia ciąg znaków będący poleceniem dla serwera, zapisuje go do kolejki FIFO `pescador.in` oraz odczytuje kolejne linie wyników z kolejki FIFO `pescador.out` i wypisuje je na ekranie. Ponieważ operacje na kolejkach FIFO mają charakter blokujący, w przypadku niedziałania serwera klient będzie czekał na jego uruchomienie. Skrypt wymaga istnienia interpretera języka Perl `/usr/bin/perl`. Oczywiście w razie potrzeby, zmieniając pierwszą linię skryptu, można zmienić ścieżkę do interpretera.

12.1.3. Konwerter informacji morfologicznej z analizatora SAM

Konwerter `samimp.pl` jest skryptem napisanym w języku Perl. Pobiera dane wejściowe z pliku `wyniki.sam`, tworzonych przez SAM-a podczas pracy w trybie wsadowym. Wyniki wypisuje na standardowe wyjście, więc należy je przekierować do odpowiedniego pliku. Konwerter w jednym przebiegu analizuje kolejne linie pliku wejściowego, tłumacząc je na odpowiednie linie w formacie roboczym Pescadora, które wypisuje na standardowe wyjście. Szersze komentarze techniczne znajdują się w samym skrypcie. Skrypt wymaga interpretera języka Perl `/usr/bin/perl`. Oczywiście w razie potrzeby, zmieniając pierwszą linię skryptu, można zmienić ścieżkę do interpretera.

12.1.4. Konwerter zerowej informacji morfologicznej

Konwerter `zeroimp.pl` jest skryptem napisanym w języku Perl. Pobiera dane wejściowe wprost z pliku tekstowego, podanego w pierwszym parametrze wywołania. Wyniki wypisuje na standardowe wyjście, więc należy je przekierować do odpowiedniego pliku. Konwerter w jednym przebiegu analizuje kolejne linie pliku wejściowego, tłumacząc je na odpowiednie linie w formacie roboczym Pescadora, które wypisuje na standardowe wyjście. Z braku takich danych na wejściu, wyniki oczywiście nie zawierają interpretacji morfologicznych (XML-owych elementów `<int>`). Za wyrazy morfologiczne uznawane są wyrazy alfabetyczne. Skrypt wymaga interpretera języka Perl `/usr/bin/perl`. Oczywiście w razie potrzeby, zmieniając pierwszą linię skryptu, można zmienić ścieżkę do interpretera.

12.2. Niektóre algorytmy

W tym podrozdziale przedstawię zarys bardziej interesujących algorytmów, zastosowanych w kodzie serwera.

12.2.1. Wyszukiwanie

Za wyszukiwanie odpowiedzialna jest rekurencyjnie wywoływana funkcja `eat`. Stara się ona odnaleźć w ciągu wyrazów pierwsze wystąpienie segmentu pasującego do zapytania. Parametry jej wywołania:

1. dotąd dopasowane – `Tword list`. Lista wyrazów dopasowanych do danego momentu. Jeśli parametr 3. jest już pusty, ta lista jest ostatecznym wynikiem działania funkcji.
2. lista wyrazów na której mamy szukać wzorca – `Tword list`. Dotąd nieprzejrany sufix pierwotnych danych wejściowych. Jeśli pierwszy element parametru 3. uda się dopasować do jego prefiksu, funkcja zostanie wywołana rekurencyjnie, przy czym ów prefiks zostanie w nowym wywołaniu przeniesiony z parametru 2. do 1., parametr 3. skrócony o dopasowany element, a parametr 4. ustawiony na `FINDBEG`.
3. lista definicji poszukiwanego fragmentu, którą należy dopasować – `Tqworddef list`. Dotąd niedopasowany fragment zapytania.
4. czy jesteśmy w środku wzorca – `{FINDANY,FINDBEG}`. Informacja o tym, czy jesteśmy w środku listy definicji poszukiwanego fragmentu. Funkcja `eat` ma odnaleźć pierwsze wystąpienie segmentu pasującego do zapytania; jeśli pierwszy element list z parametru 2. nie pasuje do pierwszego elementu listy z parametru 3., należy wywołać funkcję

rekurencyjnie, z parametrem 2. skróconym o pierwszy element – czyli po prostu próbować tego samego zapytania dalej. Jednak jeśli już jest obsługiwane głębsze wywołanie rekurencyjne `eat`, czyli jakieś elementy tekstu zostały już dopasowane, następnych nie można szukać gdziekolwiek, tylko bezpośrednio po tych dopasowanych. Parametr 4. reguluje więc zachowanie w przypadku porażki dopasowania: czy już nie ma szans, czy należy próbować dalej.

5. Tablica symboli (przypisania zmiennych) – `Tsymbol list`. Dla każdej ze zmiennych przechowywana jest lista ciągów, reprezentujących wartości, jakie może ona przyjmować. Więcej na ten temat w punkcie 12.2.2.

Funkcja `eat()` zwraca piątkę taką samą pod względem typów, lecz uaktualnioną: elementy dopasowane do początku wzorca (parametr 3.) są przeniesione z początku przeszukiwanego tekstu (parametr 2.) na koniec tekstu dotąd dopasowanego (parametr 1.). Elementy przeszukiwanego tekstu, które na pewno nie pasują do początku wzorca, są likwidowane.

Może się zdarzyć tak, że `eat` zacznie dopasowywać wzorzec, dopasuje jeden lub więcej segmentów, przejdzie w tryb `FINDBEG` i w pewnym momencie dalszy ciąg wzorca nie będzie już pasował. Rekurencyjna konstrukcja funkcji powoduje, że w takim przypadku również pierwotne wywołanie `eat` (w trybie `FINDANY`) zwróci błąd. Nie oznacza to jednak, że wzorzec w tekście nie występuje. W takim wypadku należy wywołać funkcję ponownie, z tym samym wzorcem co poprzednio, na innym fragmencie wejściowej listy wyrazów (parametr 2.).

Wywołanie na ogonie pierwotnej listy działa prawidłowo, ale bardzo nieefektywnie. Wyobraźmy sobie bowiem ciąg wyrazów

jeden i dwa i trzy i cztery i pięć i sześć i siedem i osiem i sześć i zero.

oraz zapytanie

sześć i zero.

Funkcja `eat` dopasuje pierwszy element zapytania i wywoła się ponownie:

```
eat([sześć],[i siedem i osiem i sześć i zero],[i zero],FINDBEG,[])
```

dopasuje drugi element zapytania i dalej wywoła się:

```
eat([sześć i],[siedem i osiem i sześć i zero],[zero],FINDBEG,[])
```

Teraz oczywiście poniesie porażkę, ponieważ *siedem* nie pasuje do *zero*. Zwrócony zostanie wynik:

```
([sześć i],[siedem i osiem i sześć i zero],[ERROR('just')],FINDBEG,[]).
```

W tym momencie należy próbować dalej. Łatwo zauważyć, że kolejne wywołanie na ogonie listy wejściowej *i dwa i trzy ...* odniesie podobny skutek, tak samo dla kolejnego ogona *dwa i trzy ...*. Zauważmy, że już pierwsze wywołanie `eat` przechodziło tę listę od początku i udało się stwierdzić, że do momentu pierwszego napotkania elementu *sześć* nie da się nic dopasować; każde następne powtarzałoby te same porównania niepotrzebnie.

W przykładzie widać, że kolejną sensowną próbę należy podjąć na liście *i siedem i osiem i sześć i zero*. Stanowi ona ogon listy powstałej z połączenia 1. i 2. parametru wyniku otrzymanego w pierwotnym wywołaniu `eat`.

Nie można po prostu użyć listy z parametru 2. Wtedy w przypadku bardziej ogólnego zapytania, pasującego zarówno do ciągu *sześć i zero*, jak i do *i siedem i*, ten drugi wynik byłby pominięty, gdyż kolejna próba wyszukiwania dokonana byłaby na ciągu rozpoczynającym się od *siedem*.

Do rzeczywistego wyszukania pierwszego wystąpienia segmentu pasującego do zapytania konieczne jest zatem użycie pewnego wrappera do funkcji `eat` – funkcji `findOne`. Jej parametry to lista wyrazów, sparsowane zapytanie oraz stan zmiennych. Wywołuje ona `eat` i sprawdza wynik. Istnieją trzy możliwości, sprawdzane po kolei:

- 3. parametr jest już pusty – odnaleziono fragment pasujący do całego poszukiwanego wzorca. Sukces.
- 2. parametr jest już pusty (a 3. nie był, co zostało sprawdzone przez chwilą) – już koniec tekstu, a nadal nie odnaleziono wzorca. Porażka.
- żadne z powyższych – trzeba szukać dalej. Ponownie wywoływane jest `eat` na ogonie listy powstałej z połączenia 1. i 2. parametry wyniku wywołania pierwotnego.

By odnaleźć nie tylko pierwsze, ale wszystkie wystąpienia, funkcja `findOne` jest z kolei wywoływana przy pomocy wrappera `findAll`. Wywołuje on `findOne` i sprawdza jego wynik (trójkę). Jeśli pierwszy element trójki jest listą pustą, zwraca listę pustą (nic nie znaleziono). Jeśli nie, zwraca listę złożoną z niego oraz wyniku `findAll` na dotąd nie przetworzonym końcu listy wyrazów, zwróconym przez `findOne`.

Alternatywa

Gdy pierwszy element listy w 3. parametrze jest postaci `ALTER(qwd1, qwd2)::pattrest`, czyli zadano zapytanie `OR(qwd1, qwd2)`, próbujemy najpierw dopasować (wykonując `eat`) do tekstu wzorzec postaci `qwd1::pattrest` oraz, niezależnie, wzorzec `qwd2::pattrest`. Są 4 możliwe konfiguracje wyników, rozróżniane na podstawie trzeciego elementu zwracanych przez `eat` piątek:

- dwie listy puste – dopasowano obie możliwości `qwd1` i `qwd2`. Zwracamy ten wynik, którego początek jest bliższy początkowi wejściowej listy wyrazów. Drugi wynik zostanie odnaleziony ponownie przy kolejnym wywołaniu `findOne`. Nie wpływa to na liniowość czasu przeszukiwania, a pozwala zachować spójność algorytmu. W przypadku dwóch wyników z początkiem w tym samym miejscu wybierany jest dłuższy; wynik krótszy nie będzie już odnaleziony (podejście zachłanne).
- pierwsza lista pusta, druga nie – dopasowano `qwd1`. Zwracamy wynik dla `qwd1`.
- druga lista pusta, pierwsza nie – dopasowano `qwd2`. Zwracamy wynik dla `qwd2`.
- żadna lista nie jest pusta – nic nie dopasowano. Jeśli 4. parametr na to zezwala, przeszukujemy dalej na dłuższej z list wyrazów będących 2. parametrem wyników pierwotnych prób (dla `qwd1::pattrest` i `qwd2::pattrest`), przy czym wybieramy dłuższą, by nie przeskoczyć żadnego wystąpienia. Jeśli 4. parametr to `FINDBEG`, meldujemy porażkę wyszukiwania.

Koniunkcja

Gdy pierwszy element listy w 3. parametrze jest postaci `CONJ(qwd1, qwd2)::pattrest`, czyli zadano zapytanie `AND(qwd1, qwd2)`, sytuacja jest nieco prostsza. Algorytm próbuje dopasować (wykonując `eat`) do tekstu wzorzec postaci `qwd1::pattrest`. Jeśli się nie uda, to melduje porażkę - wywoływany `eat` musiał bowiem nie znaleźć nic pasującego aż do końca danych wejściowych, więc nie ma sensu próbować dalej. Jeśli się uda, to wywoływany jest `eat` dla wzorca postaci `qwd2::pattrest`, ale nie na tym samym tekście wejściowym co poprzednio. Drugi człon koniunkcji należy bowiem dopasować w dokładnie tym samym miejscu, co pierwszy. Dlatego drugą próbę podejmujemy na sumie 1. i 2. parametru wyniku próby pierwszej, czyli sumie ciągu odnalezionego dla `qwd1` i dotychczas nieprzejrzanego fragmentu danych wejściowych, a czwarty parametr ustawiany jest na `FINDBEG`.

Jeśli druga próba się nie powiedzie i wywołanie było w trybie FINDANY, poszukiwania całego wzorca `CONJ(qwd1,qwd2)::pattrest` kontynuowane są na danych wejściowych skróconych o te elementy, na których na pewno nie ma nic pasującego do `qwd1`, co stwierdziliśmy w pierwszej próbie. W trybie FINDBEG porażka drugiej próby oznacza porażkę całego poszukiwania.

Powtórzenie

Gdy pierwszy element listy w 3. parametrze jest postaci `REPEAT(qwd,lo,hi)::pattrest`, czyli zadano zapytanie `REP(qwd,n,k)`, działanie zależy od parametrów `n` i `k` (górnego i dolnego ograniczenia liczby powtórzeń). Po kolei sprawdzane są następujące możliwości:

- `(0, 0)` – powtórzenie od 0 do 0 razy, czyli ten element zapytania należy zignorować. Podejmowana jest próba dopasowania tego samego tekstu wejściowego do reszty wzorca.
- `(0, k)` – powtórzenie od 0 do `k` razy. Wyszukiwanie powtórzeń działa niezachłannie, więc jeśli uda się dopasować powtórzenie 0 razy (w sposób opisany powyżej), to zwracany jest wynik tegoż dopasowania. W przeciwnym przypadku zwracany jest wynik dla powtórzeń od 1 do `k` razy. W przypadku potrzeby dopasowywania zachłannego, program należy w tym miejscu zmodyfikować tak, by wypróbował kolejne liczby powtórzeń aż do porażki.
- `((n, k)` – powtórzenie od `n` do `k` razy. Dokonywana jest podmiana elementów zapytania na przodzie listy z parametru 3. i zwracana odpowiedź dla `eat` wykonanego z tą podmienioną listą. Algorytm sprawdza, czy najpierw występuje `n` powtórzeń `qwd`, a zaraz po nim od 0 do `(k-n)` powtórzeń. Pierwsze sprawdzenie dokonywane jest przy użyciu dodatkowego typu elementu zapytania - dopasowania dokładnie zadanej liczby powtórzeń `REPEATEXACT`. Tam również dokonywana jest podmiana elementów zapytania, a mianowicie `REPEATEXACT(qwd,n)` zamieniane jest na `qwd::qwd::...::qwd`, gdzie `qwd` występuje dokładnie `n` razy.

12.2.2. Uzgadnianie zmiennych

Podczas wyszukiwania program czyta kolejne elementy sparsowanego zapytania `q` i sprawdza, czy kolejne elementy tekstu `e` do nich pasują. Jeśli przy tym w zapytaniu jako żądana wartość kategorii `k` napotkana zostanie zmienna `x`, obowiązuje następująca procedura:

- jeśli zmienna `x` nie została dotąd przypisana
Przypisz na nią listę wszystkich możliwych wartości kategorii `k` (we wszystkich interpretacjach `e`). Uznaj, że dopasowane prawidłowo i szukaj następnego elementu zapytania `q`, z uaktualnioną tablicą wartości zmiennych.
- jeśli na zmienną przypisana jest lista wartości `l`
Oblicz przecięcie `l` oraz listy wszystkich możliwych wartości kategorii `k` (we wszystkich interpretacjach `e`). Jeśli przecięcie jest puste, uznaj że nie udało się dopasować prawidłowo i szukaj `q` od początku w dalszej części tekstu, z pustą tablicą wartości zmiennych. Jeśli przecięcie niepuste, zapisz jego wartość na zmienną `x`, uznaj, że dopasowane prawidłowo i szukaj następnego elementu zapytania `q`, z uaktualnioną tablicą wartości zmiennych.

Tablica symboli jest listą elementów typu `Tsymbol = SYMB of int * string list`. Jako zmienna nieprzypisana traktowana jest zmienna, której numeru nie ma nigdzie na liście

w pierwszym polu oraz zmienna, która znajduje się na liście, ale w drugim polu ma pustą listę stringów.

12.2.3. Wyszukiwanie w wynikach

Dzięki wygodnej obsłudze struktur danych, wyszukiwanie w wynikach udało się zaimplementować niewielkim kosztem. Zrealizowany pomysł polega na tym, by wyniki zapytania (listę list wyrazów - `Tword list list`) konwertować w razie potrzeby na taką samą strukturę, do jakiej parsowane są pliki robocze. Każdy wynik (lista wyrazów) tłumaczony jest na jedną strukturę `Tline`, czyli na jedną linię logiczną. Linie logiczne zawierane są w strukturze `Tfile`, identyczną z powstającą po wczytaniu pliku roboczego. Całe wyszukiwanie na wynikach z punktu widzenia funkcji wyszukiwujących przebiega więc analogicznie jak na całości pliku.

Rozdział 13

Możliwości dalszego rozwoju systemu

1. Dostosowanie modułu obsługi wyrażeń regularnych. Użyty moduł nie obsługuje kodowań innych niż ośmiobajtowe, co ogranicza uniwersalność językową systemu z racji uniemożliwienia obsługi korpusów zakodowanych zgodnie z Unicode. Poza tym w wyrażeniach regularnych nie są prawidłowo obsługiwane symbole `^` i `$` wyznaczające początek i koniec wyrazu. Wymaga to zmodyfikowania istniejącego kodu w SML lub napisania modułu zupełnie od nowa.
2. Opcja wypisywania stanu przypisań zmiennych po wykonaniu wyszukiwania. Wykonanie zapytania
`[pos='adj' word=#1] rower`
dawałoby na zmiennej `#1` listę wszystkich przymiotników, jakie się pojawiły przez wyrazem *rower*, a
`[lemma='zielony' case=#1]`
przypisywałoby na `#1` listę wszystkich przypadków, w jakich wystąpił wyraz *zielony*.
3. Stworzenie nowych modułów konwersji informacji morfologicznej, ze wskazaniem opracowanego w IPI PAN analizatora *Morfeusz*.

Dodatek A

Opis dołączonej płyty z oprogramowaniem

W głównym katalogu płyty znajduje się plik PDF z niniejszym tekstem oraz podkatalogi `pescador` i `source`.

W katalogu `pescador` znajduje się skompilowany, gotów do użycia program oraz konwertery wejściowe. Całość, dla zachowania praw dostępu oraz umożliwienia przechowania na płycie plików specjalnych, została spakowana programem `tar`. Aby odpakować archiwum, należy wydać polecenie `tar xvzf pescador10.tgz`.

W katalogu `source`, w osobnych podkatalogach, znajdują się źródła texowe niniejszej pracy oraz źródła serwera `pescador`.

Dodatek B

Korespondencja z Radą Języka Polskiego

Cytowana niżej korespondencja miała na celu ustalenie właściwego terminu na określenie programu tworzącego konkordancje.

Oto mój pierwszy e-mail, wysłany 14 listopada 2003:

Dzień dobry!

Piszę pracę magisterską, której główną częścią jest program komputerowy tworzący konkordancje. Naturalna jest konieczność opatrzenia go jakąś nazwą.

Przychodzą mi do głowy dwie: konkordanser i konkordancer, przy czym pierwsza intuicyjnie lepiej mi brzmi, a druga chyba jest poprawniejsza etymologicznie.

Którą z nich powinienem przyjąć? A może żadnej z nich?

Z góry dziękuję za pomoc,
Łukasz Degórski

Po pewnym czasie (22 listopada) otrzymałem odpowiedź:

Warszawa, dn. 22.11.03
RJP- 436/W/2003

Szanowny Panie,
przewodniczący Komisji Terminologii informatycznej, prof. Andrzej Jacek Blikle, prosi Pana o opisanie w kilku zdaniach efektu działania programu, o którym Pan pisze. Jest to niezbędne do udzielenia odpowiedzi.

Z wyrazami szacunku
Katarzyna Kłosińska

Zwlekałem z odpowiedzią i 10 stycznia 2004 roku otrzymałem przypomnienie:

Warszawa, dn. 10 stycznia 2004 roku
RJP-7/W/2004

Szanowny Panie!

Uprzejmie przypominam swą prośbę, skierowaną do Pana w liście z dnia 22 listopada 2003 roku, o opisanie w kilku zdaniach efektu działania programu, o którym Pan pisze. Jest to niezbędne do tego, by Komisja Terminologii Informatycznej RJP mogła udzielić Panu odpowiedzi na Pański list z dn. 14 listopada 2003 roku.

Jeśli Pańska prośba o poradę w sprawie terminu informatycznego jest już nieaktualna, to prosimy o poinformowanie nas o tym.

Odpowiedziałem 22 stycznia:

Program ów ma na wejściu tekst (otagowany, jeśli ma to jakieś znaczenie). Użytkownik zadaje mu zapytanie w postaci definicji żadanego słowa lub listy żądanych słów. Program wyszukuje w tekście wystąpienia żadanego słowa lub listy oraz wypisuje je, wraz z pewnym kontekstem wystąpienia. Jest to tak zwana konkordancja.

Na tym korespondencja się urwała.

Dodatek C

GNU Free Documentation License

Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one

or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under

the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Dodatek D

Licencja na używanie modułów pochodzących z NJ SML

Do obsługi wyrażeń regularnych zostały użyte moduły dystrybuowane z kompilatorem New Jersey SML. Zgodnie z warunkami licencji, jestem zobowiązany zawrzeć poniższe, dotyczące ich, informacje w dokumentacji, którą jest niniejsza praca.

Copyright (c) 1989-1998 by Lucent Technologies

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent Technologies, Bell Labs or any Lucent entity not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

Lucent disclaims all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall Lucent be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.”

Prace cytowane

- [1] Piotr Bański, *The proposed encoding scheme for the IPI PAN corpus*, Prace IPI PAN **936** (December 2001).
- [2] Janusz S. Bień, *O pojęciu wyrazu morfologicznego*, w: Włodzimierz Gruszczyński, Urszula Andrejewicz, Mirosław Bańko, Dorota Kopcińska (red.), *Nie bez znaczenia ...* (2001), s. 67–77.
- [3] ———, *Wyrazy morfologiczne i morfosyntaktyczne w praktyce*, w: Andrzej Moroz, Marek Wiśniewski (red.), *Studia z gramatyki i semantyki języka polskiego* (2004), s. 171–182.
- [4] Rafał T. Prinke, *Terra rubrica – terra electronica. Najkrótsza historia metatekstu*, w: Nuntius Vestustatis – Prace ofiarowane Profesorowi Jerzemu Wiślockiemu, pod red. Adama Bieniaszewskiego i Rafała T. Prinke. Wersja 1.0.0 z dnia 12.10.1998, widziana ostatnio 3.09.2004 pod adresem <http://www.bkpan.poznan.pl/biblioteka/JW70/terra.htm>.
- [5] Adam Przepiórkowski, *Korpus IPI PAN. Wersja wstępna.*, Instytut Podstaw Informatyki PAN, Warszawa, 2004.
- [6] Adam Przepiórkowski, Marcin Woliński, *The Unbearable Lightness of Tagging. A Case Study in Morphosyntactic Tagging of Polish*, Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03), EACL 2003., Praca dostępna też elektronicznie pod adresem <http://dach.ipipan.waw.pl/adamp/Papers/2003-eacl-ws03/ws03.pdf>.
- [7] Jan Tokarski, *Schematyczny indeks a tergo polskich form wyrazowych*, Wydawnictwo Naukowe PWN, Warszawa, 2002.
- [8] Marcin Woliński, *System znaczników morfosyntaktycznych w korpusie IPI PAN*, Polonica **XXII–XXIII** (2003), s. 39–55. Praca dostępna też elektronicznie pod adresem <http://dach.ipipan.waw.pl/CORPUS/znakowanie.pdf>.
- [9] *Český národní korpus. Úvod a příručka uživatele*, Filozofická Fakulta UK, Praha, 2000.