**Uniwersytet im. Adama Mickiewicza**

**Marcin Walas**
nr albumu: 287572

# Pozyskiwanie wiedzy z bazy artykułów/tekstów za pomocą aplikacji typu *wyszukiwarki internetowej* odpowiadającej na pytania o czas i miejsce

Praca magisterska na kierunku:
INFORMATYKA
Promotor:
**dr hab. Krzysztof Jassem**

Poznań 2009

# Oświadczenie

Ja, niżej podpisany **Marcin Walas** student Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt.:

**Pozyskiwanie wiedzy z bazy artykułów/tekstów za pomocą aplikacji typu *wyszukiwarki internetowej* odpowiadającej na pytania o czas i miejsce**

napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem tej rozprawy lub jej części od innych osób.

Oświadczam również, że egzemplarz pracy dyplomowej w formie wydruku komputerowego jest zgodny z egzemplarzem pracy dyplomowej w formie elektronicznej.

Jednocześnie przyjmuję do wiadomości, że gdyby powyższe oświadczenie okazało się nieprawdziwe, decyzja o wydaniu mi dyplomu zostanie cofnięta.

| ..... | ..... |
|---|---|
| data | podpis |

## Streszczenie

Praca opisuje proces opracowania inteligentnej wyszukiwarki, która potrafiłaby trafnie odpowiedzieć na zadane przez użytkownika pytanie w języku polskim. Odpowiedź powinna być uzyskana na podstawie wiedzy zawartej w dowolnej bazie tekstów niesformatowanych (*plain-text*) i być możliwie najbardziej szczegółowa i trafna. W pracy opisuję system bardziej zaawansowany od istniejących w chwili obecnej systemów tego typu, które bazują jedynie na wyszukiwaniu słów kluczowych, pozostawiając użytkownikowi często żmudną pracę dotarcia do konkretnej odpowiedzi. Przykładowym pytaniem zadanym aplikacji może być np. „Gdzie odbędzie się koncert zespołu X?". Odpowiedź, jakiej mógłby udzielić system to np. „Poznań - Malta". Ze względu na szeroki zakres rozpatrywanej dziedziny, w pracy ograniczę się tylko do przedstawienia problemu odpowiedzi na pytania o czas i miejsce.

## Słowa kluczowe

question answering, spatial and temporal question answering, natural language processing, information extraction, information retrieval, named entity recognition, web crawler

# Spis treści

# Wstęp

Pozyskiwanie wiedzy to proces polegający na wyszukaniu i udostępnieniu informacji. Źródłem wiedzy może być dowolne medium będące nośnikiem informacji, np. baza danych, kolekcja dokumentów tekstowych lub zbiór stron internetowych. Pozyskiwanie wiedzy jest elementem takich systemów informatycznych jak: wyszukiwarki internetowe (np. *Google*), wyszukiwarki w katalogach (np. system wyszukiwania pozycji bibliotecznych), wirtualni asystenci (np. asystent Microsoft Office) czy systemy eksperckie.

Niniejsza praca magisterska dotyczy problemu pozyskiwania wiedzy z kolekcji dokumentów zapisanych w postaci tekstu niesformatowanego (ang. *plain-text*) i znajdujących się w sieci internet. Obecnie do przeszukiwania internetu wykorzystuje się zwykle wyszukiwarki internetowe (np. *Google*, *Yahoo!*). Systemy takie wyszukują informacje przyjmując zapytania w formie sztucznego języka *słów kluczowych*. Jako odpowiedź zwracają listę odnośników (*linków*) do stron internetowych zawierających szukane informacje. Rozwiązanie takie posiada szereg wad, takich jak: trudność i nienaturalność języka zapytań, zrzucenie ciężaru wyszukania konkretnej informacji na użytkownika (któremu faktycznie wyświetlane są jedynie źródła informacji, a nie same informacje), niemożność zastosowania w pewnych nowoczesnych koncepcjach aplikacji (np. w aplikacjach typu *chatter-bot*, prowadzących dialog z użytkownikiem).

W pracy przedstawiam koncepcję Question Answering Systems (systemów odpowiadania na pytania, w skrócie systemów QA), której podstawowym zadaniem jest wyeliminowanie wymienionych powyżej wad tradycyjnych systemów wyszukujących informacje. Systemy QA przyjmują zapytanie w postaci zdania w języku naturalnym oraz wyświetlają odpowiedź będącą możliwie krótkim fragmentem tekstu.

Celem pracy jest zbudowanie systemu QA działającego w języku polskim. System ten, o nazwie **Hipisek.pl**, ma postać wyszukiwarki internetowej, która zamiast tradycyjnego zapytania, w postaci listy słów kluczowych, przyjmuje pytanie zadane w języku polskim. Odpowiedzią na pytanie jest fragment tekstu zawierający szukane przez użytkownika informacje. Źródłem wiedzy dla systemu jest kolekcja plików tekstowych pozyskiwanych automatycznie z internetu. Ze względu na ogromną różnorodność problemów rozpatrywanych w ramach dziedziny systemów QA, w pracy ograniczę się do omówienia zagadnienia odpowiadania na pytania o czas i miejsce dotyczących wiedzy zawartej w internetowych serwisach prasowych. Ważnym elementem źródła wiedzy systemu jest jego zmienność (baza wiedzy rośnie wraz z pojawianiem się nowych artykułów na stronach serwisu prasowego).

W pierwszym rozdziale pracy omawiam dziedzinę systemów QA. W rozdziale 2. opisuję proces zbierania materiału korpusowego, niezbędnego do rozważań nad dziedziną QA, oraz przedstawiam model teoretyczny zbudowanego systemu **Hipisek.pl**. W rozdziale 3. przedstawiam formalizację podstawowych pojęć dotyczących procesu odpowiadania na pytanie oraz definiuję komputerowy formalizm reprezentacji pytania za pomocą zapytań *QQuery*. Rozdział 4 zawiera opis analizy i zrozumienia pytania w systemie **Hipisek.pl**. W rozdziale 5 przedstawiam sposób organizacji bazy wiedzy projektu oraz metody wyszukiwania potencjalnych odpowiedzi spośród zapisanych w bazie artykułów. Opis metod ekstrakcji konkretnych odpowiedzi z wyszukanych kandydatów znajduje się w rozdziale 6. Rozdział 7 zawiera opis szczegółów technicznych zbudowanego systemu, jego ewaluację oraz krótką listę możliwych punktów rozwoju projektu.

Dodatki do pracy zawierają opisy formalizmów zbudowanych dla potrzeb systemu **Hipisek.pl**. Dodatek A zawiera opis gramatyki zapytań QQuery. Dodatek B zawiera opis gramatyki szablonów pytań. Dodatek C zawiera opis gramatyki modułu rozpoznającego jednostki nazwane. W dodatku D umieściłem pytania zestawu testowego, używanego w procesie ewaluacji serwisu.

Zbudowany w ramach pracy system został wdrożony i udostępniony pod adresem http://www.hipisek.pl. Jest to, w chwili oddania pracy do druku, jeden z niewielu systemów tego typu, działający w języku polskim.

ROZDZIAŁ 1

# Wprowadzenie

*„Nadmiar wiedzy jest równie szkodliwy jak jej brak."*
**Emil Zola**

W rozdziale opisuję w skrócie problemy związane z wyszukiwaniem informacji, które powstają w wyniku bardzo szybkiej ekspansji internetu. Przedstawiam dziedzinę informatyki nazywaną w literaturze *Question Answering* (QA), której jednym z zagadnień jest temat niniejszej pracy. Rozdział kończę krótkim opisem istniejących komercyjnych serwisów QA, projektów badawczych QA oraz przedstawieniem własnego systemu QA – **Hipisek.pl**.

## 1.1. Rozwój rynku wyszukiwarek internetowych

Według danych opublikowanych na stronie **hypertextbook.com** wielkość internetu w roku 1997 szacuje się na ok. 300 milionów stron[1]. Według tego samego źródła, w roku 2002 było to już **8 miliardów** stron. Tak wielka ilość informacji oprócz pozytywnych skutków, stwarza także pewne problemy:

- Znalezienie konkretnej informacji jest w tak ogromnym medium bardzo trudne (należy przeszukać niesłychanie dużą ilość stron),
- Niezwykle trudno stworzyć coś w rodzaju *spisu treści internetu* (jest to zadanie tak abstrakcyjne, że aż śmieszne), przez co istnieje duża szansa, że wiele mniejszych witryn, nie prowadzących agresywnej polityki marketingowej, nie zaistnieje w sieci, gdyż żaden z użytkowników nie dowie się o ich istnieniu,
- Powstające katalogi stron internetowych (np. *Open Directory Project* www.dmoz.org) nie nadążają w dodawaniu kolejnych witryn.

Te oraz inne problemy spowodowały ogromny wzrost popularności **wyszukiwarek internetowych**, których podstawowym zadaniem jest wyświetlenie adresów url (potocznie nazywanych *linkami*) do stron, które spełniają **potrzebę informacyjną** użytkownika. Potrzeba informacyjna reprezentowana jest za pomocą zapytania w postaci listy słów kluczowych, które powinny znaleźć się na danej stronie. Dawid Weiss w artykule [1] słusznie zauważa, że zapytanie takie przypomina język sienkiewiczowskiego Kalego. Użytkownik, który szuka informacji o najbliższych koncertach w Poznaniu, używając chociażby popularnej wyszukiwarki *Google* może wszakże wyszukiwać korzystając z zapytań w naturalnym dla niego języku:

> Wyświetl mi listę najbliższych koncertów w Poznaniu.

Jednak najprawdopodobniej nie otrzyma satysfakcjonującej go odpowiedzi. Jeśli chce poprawić otrzymane wyniki, zmuszony jest do zadania zapytania, które mogłoby wyglądać na przykład tak:

[1] *„The Physics Factbook"*, dane według *NEC Research* http://hypertextbook.com/facts/2007/LorantLee.shtml

koncerty Poznań

Niestety odpowiedź jaką otrzyma pozostawia wiele do życzenia, ponieważ wyświetlone zostaną tylko *linki* do stron, które mogą (ale nie **muszą**!) zawierać poszukiwane informacje. Użytkownik będzie musiał wykonać żmudną pracę przeglądania zawartości całej strony, by odnaleźć interesujący go fragment. Powoduje to również powstanie wielu innych problemów oraz niedogodności np. problem z obsługą polskiej fleksji (według artykułu [1] Google w 2002 roku nie potrafił odmieniać polskich wyrazów, obecnie wydaje się, że funkcjonalność taka została zaimplementowana[2]) czy problem związany z adekwatnością prezentowanego zapytania do potrzeby informacyjnej użytkownika (użytkownik przez ograniczenia języka zapytań, bądź własną niewiedzę mógł nie w pełni wyrazić problem, który chciał rozwiązać).

Pomimo tych niedogodności wyszukiwarki internetowe odnoszą spory sukces, a pozycja *Google* pośród nich jest bardzo silna. Według danych zawartych w książce Witolda Abramowicza [2] udział wyszukiwarek w rynku przedstawia się następująco (dane na maj 2007 r.):

1. **Google** – 50,7%
2. Yahoo! – 25,4%
3. Microsoft – 10,3%
4. Ask – 5,0%
5. Time Warner Network – 4,6%
6. Inne – 3,0%

Pojawiają się jednak nowe koncepcje wyszukiwarek, w których stosuje się bardziej intuicyjne metody zadawania pytań i podawania odpowiedzi. Jedną z takich koncepcji są *Question Answering Systems* (Systemy Odpowiadania na Pytania) zwane w skrócie QA Systems (Systemami QA).

## 1.2. Definicja pojęcia „Question Answering"

W artykule [3] autorzy opisują **systemy QA** jako technologię, która stara się dostarczyć bardziej intuicyjne metody dostępu do informacji, poprzez przetwarzanie zapytań zadanych w języku naturalnym i podaniu zwięzłej odpowiedzi. Z kolei w pracy [4] zawarta jest następująca definicja[3]:

> **Definicja 1.1** ***Question Answering (QA)*** *to proces interaktywny, którego celem jest dostarczenie precyzyjnej odpowiedzi na zapytanie zadane w języku naturalnym.*

[2] Niestety *Google* ze względu na politykę firmy udostępnia bardzo skąpe informacje dotyczące tworzonych przez nią produktów, przez co można tylko przypuszczać (na podstawie obserwacji wyników), że tak jest w istocie.

[3] Junlan Feng, *Question Answering with Question Answer Pairs on the Web*, tłumaczenie własne.

Pojęcie QA często łączy się z dwoma innymi technikami wyszukiwania informacji: **Information Retrieval** (IR, wyszukiwanie informacji) oraz **Information Extraction** (IE, ekstrakcja informacji).

W książce [5] zawarto następującą definicję[4]:

> **Definicja 1.2** ***Information retrieval (IR)*** *to znalezienie materiału (zwykle dokumentów) o nieustrukturyzowanej naturze (zwykle w postaci tekstu), który zaspokaja potrzebę informacyjną, spośród dużych kolekcji danych (zwykle zapisanych w komputerach).*

Natomiast w artykule [6] przez **Information Extraction** rozumie się[5]:

> **Definicja 1.3** ***Information Extraction*** *dotyczy automatycznej ekstrakcji ustrukturyzowanej informacji w postaci jednostek, relacji pomiędzy jednostkami i własności opisujących jednostki, ze źródeł nieustrukturyzowanych.*

Często pojęcie **systemów QA** rozumiane jest jako kompilacja technik IE oraz IR. Podejście takie reprezentuje na przykład Zygmunt Vetulani w pracy [7].

## 1.3. TREC

Na świecie działa wiele ośrodków naukowych, zajmujących się zagadnieniem wyszukiwania i ekstrakcji informacji oraz organizowane są coraz częściej konferencje skupiające inżynierów i naukowców zainteresowanych tematem. Jedną z takich konferencji jest **TREC** (Text REtrieval Conference). Na stronie głównej projektu możemy znaleźć informację, że:

> Text REtrival Conference(TREC) współsponsorowana przez *National Institute of Standards and Technology* (NIST) i Ministerstwo Obrony Stanów Zjednoczonych powstało w 1992 roku jako część programu *TIPSTER*. Zadaniem konferencji było wspieranie badań w dziedzinie wyszukiwania informacji poprzez dostarczenie infrastruktury niezbędnej do ewaluacji na wielką skalę metodologii wyszukiwania tekstu.[6]

Obszary badań wspieranych w ramach **TREC** grupowane są w obrębie **torów** (ang. track). Jednym z *TREC tracks* jest tor dotyczący zagadnienia *Question Answering*. W ramach toru **TREC** udostępnia szereg publikacji dotyczących zagadnienia, prowadzi ranking zgłoszonych systemów QA oraz dostarcza dane testowe do ewaluacji gotowych systemów (takie jak korpusy pytań, pytania i odpowiedzi testowe, metryki oceny systemów [7]). Udostępnienie referencyjnych danych testowych umożliwia ocenę rozwiązań proponowanych przez różne ośrodki badawcze, natomiast publikowane artykuły popularyzują najlepsze koncepcje.

[4] Christopher D. Manning, Prabhakar Raghavan i Hinrich Schutze *Introduction to Information Retrieval*, tłumaczenie własne.

[5] Sunita Sarawagi, *Information Extraction*, tłumaczenie własne.

[6] Oficjalna strona TREC, http://trec.nist.gov, tłumaczenie własne.

[7] Problem zebrania korpusu pytań przedstawiam w Rozdziale 2.

## 1.4. Klasyfikacja systemów QA

W pracy Marka Maybury'ego [8] wyróżniono między innymi następujące charakterystyki i cechy systemów QA, według których można przeprowadzać ich klasyfikację[8]:

- Postać zapytania (np. lista słów kluczowych, fraz czy pełne pytanie), typ obsługiwanych pytań (np. pytania typu: kto, kiedy, co, gdzie [...]) i cel pytania (np. polecenie, komenda, informacja)[9],
- Poziom złożoności pytania i odpowiedzi,
- Charakterystyka użytkowników,
- Zamiary użytkowników,
- Charakterystyka obsługiwanych źródeł wiedzy (np. stopień niezbędnej wiedzy lingwistycznej, wiedzy o świecie),
- Wymagania dotyczące wnioskowania,
- Stopień wielojęzyczności systemu,
- Typ dostarczanych odpowiedzi (np. w postaci frazy, faktu, linku do streszczenia dokumentu),
- Interaktywność (np. reakcja na działania użytkownika).

W systemach QA często wykorzystuje się techniki i zasoby związane z przetwarzaniem wiedzy, przetwarzaniem języka naturalnego czy przechowywania i dostępu do dużych kolekcji danych, takich jak:

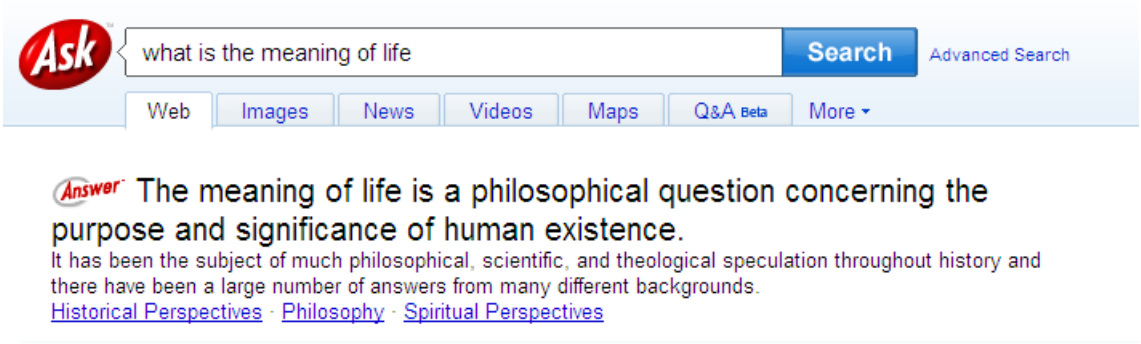
- **Ontologia** – jako formalny sposób opisu wiedzy o świecie i relacji między pojęciami[10],
- **Parsing języka naturalnego** – który dzięki analizie gramatycznej lub semantycznej zdań wykorzystywany jest często do procesu *zrozumienia pytania* oraz ekstrakcji informacji z konkretnych partii tekstu[11],
- **Rozpoznawanie jednostek nazwanych** – oznaczenie wyrazów bądź fraz o specjalnym znaczeniu (np. nazwisk, adresów, dat), często ułatwiające ekstrakcję poprawnej informacji[12],
- **Metody indeksowania dokumentów** – stosuje się np. *model wektorowy* do tworzenia indeksu dokumentu i szybkiego wyszukania konkretnych wyrazów lub fraz
- **Słowniki i tezaurusy**

[8] Mark T. Maybury, *Towards Question Answering Roadmap* (fragment), tłumaczenie własne.
[9] Więcej informacji na temat pytań i ich klasyfikacji przedstawiam w Rozdziale 3.
[10] **Ontologia** to w informatyce sposób na formalizację wiedzy.
[11] **Parsing** jest to proces analizy tekstu, którego wynikiem jest dostarczenie jego struktury gramatycznej.
[12] Pojęcie **jednostki nazwanej** zostanie przedstawione i omówione w Rozdziale 4.

## 1.5. Kierunki rozwoju QA

### 1.5.1. Komercyjne systemy QA

Pomimo coraz większego zainteresowania środowiska naukowego zagadnieniem Question Answering nie istnieje jeszcze zbyt wiele dobrze działających komercyjnych systemów QA. Do tego nielicznego grona należą serwisy **answers.com**[13], **www.answerbus.com** oraz **ask.com**. Serwisy te jednak nie obsługują języka polskiego, co jest dużym utrudnieniem dla polskojęzycznego użytkownika. Przykładowa odpowiedź na pytanie w systemie **ask.com** znajduje się na rysunku 1.1.




Rysunek 1.1: Odpowiedź na przykładowe pytanie w systemie **ask.com**

Ciekawostką jest, że obok mechanizmów typu QA, systemy komercyjne udostępniają także tradycyjne metody wyszukiwawcze (podobne do tych stosowanych m.in. przez Google). Spowodowane jest to zapewne chęcią przyciągnięcia większej ilości odbiorców oraz słabością samego mechanizmu odpowiadania na pytania.

### 1.5.2. Badawcze projekty QA

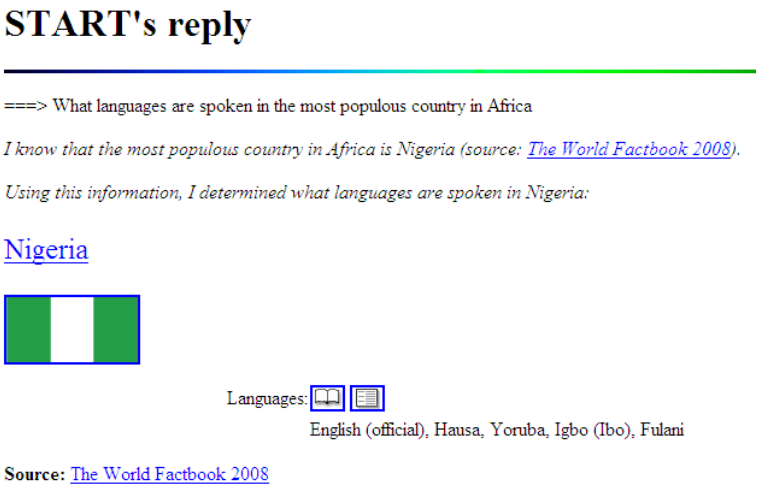
Istnieje jednak znaczna część projektów naukowych, w ramach których budowane są eksperymentalne systemy QA. Jednym z takich systemów jest **START**[14] rozwijany w *MIT Computer Science and Artificial Intelligence Laboratory*. Przykładowe pytanie wraz z odpowiedzią dostarczaną przez system znajduje się na rysunku 1.2.

Przykład ten ilustruje kilka bardzo typowych dla Question Answering problemów i ich rozwiązań:

- Podana odpowiedź jest bardzo zwięzła i dokładna, użytkownik otrzymuje dokładnie taką odpowiedź o jaką mu chodziło,
- System sam wywnioskował odpowiedź. Najpierw znalazł informację, że najbardziej zaludnionym krajem w Afryce jest Nigeria, po czym korzystając z tej wiedzy wyświetlił języki jakimi posługują się obywatele tego kraju,
- System podał źródło, z którego zaczerpnięto odpowiedź dzięki czemu użytkownik może poszukać więcej informacji na zadany temat (lub dalej odpytywać program).

[13]Tak naprawdę system ten nie jest typowym reprezentantem trendu QA, gdyż bazuje na społeczności internetowej, której członkowie ręcznie wpisują odpowiedzi na pojawiające się pytania.

[14]START Natural Language Question Answering System, http://start.csail.mit.edu/




Rysunek 1.2: Odpowiedź na przykładowe pytanie w systemie START

Możliwości systemu START są całkiem duże. Program potrafi między innymi podać bieżące informacje (np. o aktualnej pogodzie), wyświetlać szukane obrazy (np. mapy państw, obrazy artystów, fotografie znanych ludzi), opisywać znane osoby, podawać daty różnych wydarzeń (np. daty urodzeń), identyfikować relacje rodzinne, odpowiadać na pytania dotyczące historii, biografii słynnych ludzi, geografii, biologii, fizyki i wiele innych. Podobnie jednak jak przedstawione wcześniej systemy komercyjne START nie obsługuje języka polskiego.

Ostatnim z projektów, który chciałbym przedstawić w ramach niniejszej pracy jest serwis **ktoco.pl**. Jako jeden z nielicznych działających systemów QA obsługuje zapytania w języku polskim. Serwis ma obecnie status projektu badawczego i jest dopiero rozwijany. Autorzy projektu nazywają serwis *pierwszą polską metawyszukiwarką semantyczną*. Odpowiedź na zadane pytanie wyświetlana jest użytkownikowi w postaci fragmentów zaindeksowanej strony wraz z *linkiem* do strony www, z której zaczerpnięto podany fragment. Serwis ten posłużył jako projekt referencyjny w procesie ewaluacji autorskiego projektu **Hipisek.pl** (patrz Rozdział 7).

## 1.6. Hipisek.pl

W ramach niniejszej pracy stworzyłem system QA – **Hipisek.pl**. Podstawowe cechy systemu to:

- Obsługa pytań o czas i miejsce zadawanych w języku polskim,
- Źródłem wiedzy są internetowe serwisy prasowe (serwis informacyjny www.tvn24.pl oraz serwis plotkarski www.pudelek.pl),
- Interfejs systemu jest wzorowany na interfejsie wyszukiwarki *Google*,
- Odpowiedź podawana jest w postaci fragmentu tekstu, bądź kompilacji kilku fragmentów tekstów znalezionych w zaindeksowanych dokumentach,
- Istotną częścią systemu jest *robot indeksujący*, który na bieżąco pobiera zawartość pojawiających się w serwisach artykułów.

ROZDZIAŁ 2

# Model teoretyczny systemu QA

**Brak odpowiedzi jest lepszy niż zła odpowiedź...**

W rozdziale opisuję teoretyczny schemat przetwarzania QA, w ramach budowanego systemu. Opisuję również prace związane z pozyskaniem materiału lingwistycznego, niezbędnego do określenia podstawowych własności języka zadawanych pytań. W rozdziale znajdują się krótkie opisy pomysłów i zidentyfikowanych problemów, których dokładne przedstawienie nastąpi w rozdziałach kolejnych.

## 2.1. Korpus pytań

W niniejszym podrozdziale przedstawiam niezbędny materiał lingwistyczny, potrzebny do poprawnego zidentyfikowania właściwości języka, wykorzystywanego przy zadawaniu pytań. Opisuję sposób jego wykorzystania i zawartość. Konkretne przykłady zastosowania zebranego materiału opisane zostaną w kolejnych rozdziałach. Zacznę od zdefiniowania na potrzeby pracy pojęcia **korpusu**:

**Definicja 2.1** ***Korpusem*** *nazywamy dowolną próbkę (zbiór) tekstów zadanego języka.*

O ile dostępne są całkiem *porządne* źródła korpusowe dla języka angielskiego, niemieckiego czy francuskiego[1], a nawet dedykowane źródła dla dziedziny *QA* [2], to dostęp do źródeł polskich jest co najmniej trudny, a znalezienie gotowego korpusu pytań i odpowiedzi jest wręcz niemożliwe. Stąd konieczność uzyskania możliwie dużej bazy tekstów źródłowych języka polskiego, zawierających pytania (najlepiej wraz z odpowiedziami).

Pojęcie **korpusu** jest dość szerokie, dlatego na potrzeby niniejszej pracy wprowadzę jego podział na[3]:

- **Korpus zwykły** – jest to korpus składający się z dowolnych tekstów języka polskiego,
- **Ogólny korpus pytań** – jest to korpus składający się z dowolnych pytań występujących w tekście polskim,
- **Korpus pytań wyszukiwarki** – jest to korpus składający się z pytań (w języku polskim), które użytkownicy zadają wyszukiwarce internetowej.

[1] Są to np. darmowy **OPUS** dostępny na stronie *http://urd.let.rug.nl/tiedeman/OPUS/index.php* lub **Leipzig Corpus** dostępny na stronie: *http://corpora.uni-leipzig.de/*

[2] Na stronie **Text REtrieval Conference** http://trec.nist.gov/ znajduje się kilka średniej wielkości list pytań w języku angielskim, często wraz z proponowanymi odpowiedziami

[3] Ilekroć w pracy mówię o korpusie, mam na myśli korpus tekstów w języku polskim, chyba że wyraźnie zaznaczam inaczej.

### 2.1.1. Do czego potrzebny jest korpus pytań?

W projekcie zebranie korpusu pytań służy osiągnięciu trzech celów. Są to:

1. Zidentyfikowanie pytań, szczególnie często zadawanych w języku polskim, w celu ich dokładnego obsłużenia w budowanym systemie *QA*,
2. Stworzenie materiału testującego dla parsera pytań (**QParsera**) oraz gotowego serwisu odpowiadającego na pytania,
3. Określenie dziedziny *pytań o czas i miejsce* oraz zidentyfikowanie właściwości językowych zadawanych pytań.

### 2.1.2. Pozyskanie korpusu dla języka polskiego

#### Przeszukiwanie źródeł artykułów prasowych

Pierwszym sposobem na pozyskanie korpusu pytań jest wyszukanie wszelkich podejrzanych zdań w dowolnym tekście pisanym języka polskiego. Tekst taki powinien być możliwie wysokiej jakości, tzn. zawierać minimalne ilości błędów ortograficznych, typograficznych i innych. Na potrzeby przeszukiwania źródeł korpusowych przyjąłem, że[4]:

> **Definicja 2.2** ***Pytanie o czas i miejsce*** *jest to zdanie języka naturalnego, którego nadawca oczekuje uzyskania określonej w pytaniu informacji, której meritum jest czas lub miejsce, bądź w którym występuje nawiązanie do bytu czasowego lub przestrzennego.*

Definicja ta jest na tyle ogólna, by zawierać w sobie dostatecznie dużą klasę zdań języka polskiego, jednocześnie odrzucając dużą grupę zdań całkowicie zbędnych.

☞ **Uwaga** Proszę zauważyć, że w rozumieniu definicji 2.2 pytanie wcale nie musi być pytaniem w sensie gramatycznym! Na przykład zdanie oznajmujące: „Podaj datę urodzin Mieszka I." choć w sensie gramatycznym pytaniem nie jest, to w ujęciu powyższej definicji już tak. Będzie miało to kluczowe znaczenie przy formułowaniu ostatecznej definicji.

Na potrzeby pracy przyjąłem pewnego rodzaju uproszczenie w przeszukiwaniu korpusu, polegające na szukaniu tylko zdań będących pytaniami w sensie gramatycznym lub zdań oznajmujących bądź rozkazujących zaczynających się od fraz, którymi człowiek zaczyna często *zdania-pytania*, nie będące pytaniami w sensie gramatycznym. Frazy te pozyskane zostały z początkowego zrębu korpusowego zbieranego za pomocą serwisu Hipisek.pl, omówionego poniżej (około 500 zebranych pytań), oraz obserwacji korpusów dla języka angielskiego (wspomniane już źródło ze stron **TREC**) i ręcznego dodania pewnych zwrotów.

Korpusem wejściowym tekstu zwykłego stanowiło dla mnie 32 tys. artykułów prasowych, pobranych z serwisu internetowego **Gazety Wyborczej**, udostępnionego w całości z okazji święta tego dziennika i pobranych przez Mariusza Tańskiego. Teksty zostały udostępnione mi w formie tekstu HTML. Po ich oczyszczeniu (przy pomocy prostego skryptu napisanego w języku **Perl**), czyli konwersji do litego tekstu niesformatowanego, przystąpiłem do ekstrakcji pytań.

Poszukiwanie pytania przebiegało według dwóch metod:

[4] W Rozdziale 3 przedstawiam dokładniejszą i ostateczną postać niniejszej definicji. Podana tutaj definicja ma charakter poglądowy i służy jedynie do wyjaśnienia jakich pytań poszukujemy przy zbieraniu korpusu.

- Poszukiwanie pytań gramatycznych,
- Poszukiwanie zdań potencjalnie spełniających definicję 2.2.

Poszukiwanie pytań w sensie gramatycznych opierało się na zachłannym poszukiwaniu ciągów zdań spełniających poniższy wzorzec:

```
[dowolna ilość znaków innych niż ! ? . ’ " ] [pytajnik - znak ?]
```

Oczywiście jest to bardzo *naiwne* wyrażenie regularne, jednak wystarczające do opisywanego tu zadania.

Drugą z wymienionych metod zrealizowałem przy pomocy akceptacji zdań rozpoczynających się od jednej z fraz (ciągów wyrazów), potencjalnie rozpoczynających interesujące nas pytania. Listę takich fraz stworzyłem ręcznie na podstawie obserwacji pytań trafiających do serwisu **Hipisek.pl**. Do przeszukiwania korpusu posłużył program *grep* (informacje o nim zawiera np. źródło [9]) oraz skrypty języka **Perl**.

Znalezione w korpusie sekwencje poddałem oczyszczeniu[5]. Czyszczenie polegało m.in. na:

- Wyrzuceniu *„zdań”*, które nie zaczynały się wielką literą,
- Wyrzuceniu *„zdań”* zbyt długich,
- Wyrzuceniu *„zdań”* zawierających niestandardowe znaki (np. z alfabetu innego niż polski),
- Wyrzuceniu powtórzeń,
- Wyrzuceniu zbędnych białych znaków.

Po przeprowadzeniu ekstrakcji opisaną metodą, otrzymałem **10 tys.** unikatowych pytań, stanowiących pierwszy **Ogólny korpus pytań**. Oto dziesięć wybranych przeze mnie pytań z korpusu (wybór losowy):

1. Po co powstał on oraz jemu podobne w innych miejscach Szczecina?
2. Czy mówiłeś rodzicom czasami, że chcesz, żeby byli razem?
3. Dlaczego właściwie ucięły się Pana kontakty z paryską ”Kulturą”?
4. Gdzie zatem w tym człowieku rozlicznym jest sam Jerzy Ficowski?
5. Czy były senator Krzysztof Szydłowski chciał dać łapówkę strażnikom miejskim?
6. Co będzie kluczem do zwycięstwa?
7. Czego domagają się lekarze i pielęgniarki?
8. Jest Pan z tego dumny?
9. Jak pomóc innym?
10. A jak przedstawia się sprawa bramkarzy?

[5]Przez **czyszczenie** korpusu rozumiem w tym miejscu usuwanie z niego zdań zawierających błędy ortograficzne, typograficzne, nie spełniające kryteriów definicji 2.2 lub potencjalnie nie mających sensu.

### Przeszukiwanie stron internetowych

Drugim źródłem korpusu zwykłego są wszelkiego rodzaju strony internetowe, ściągane za pomocą napisanego przeze mnie (przy współpracy z dr Filipem Gralińskim) robota typu *crawler* o kodowej nazwie **alef**. Program ten przeszukiwał przez ponad dwa miesiące strony znajdujące się w katalogach internetowych: Open Directory[6] , Wirtualna Polska[7], Katalog Onet[8]. Ze stron tych wyciągał tekst „czysty", rozpoznawał język w jakim został napisany, po czym zapisywał całość na dysku. Dodatkowym źródłem tekstów internetowych było udostępnione przez dra Filipa Gralińskiego oczyszczone (tzn. pozbawione tekstów o wyjątkowo słabej jakości) archiwum list dyskusyjnych *USENET*. Korzystając z zebranych korpusów (ponad 3 miliony unikatowych stron polskich), wyekstrahowałem opisaną powyżej metodą, kolejną porcję zdań ogólnego korpusu pytań. Powstało ponad **milion** unikatowych przykładów.

Niestety przeszukiwane medium, należy do bardzo *zaśmieconych*, przez co jakość pytań jest znacznie niższa niż materiału zebranego w poprzednim źródle. Oto dziesięć pytań z korpusu internetowego (wybór losowy):

1. Nie prościej zapytać u nich?
2. Jak oni to powiększają?
3. Jakieś sugestie co do przyczyny opisanego zjawiska?
4. Naprawdę nie widzisz trendu?
5. Gdzie szukać informacji o tym samochodzie - tzn bardziej wiarygodnych od marketingowego bełkotu na oficjalnych stronach?
6. Czy jest możliwe, że komornik namierzy moje konto a jeżeli tak to jak szybko?
7. W którym miejscu porównałam się do Ciebie ?
8. I nawet nie twierdzę, że on ma rację - bo skąd mogę wiedzieć?
9. Gdzie jest Andrzej Ława.
10. To najlepiej kupić procesor który ma nadwagę czy może niedowagę ?

### Serwis Hipisek.pl

Trzecim i najważniejszym źródłem materiału korpusowego jest zrąb aplikacji do odpowiadania na pytania, powstałej w ramach niniejszej pracy magisterskiej, czyli serwisu **Hipisek.pl**.

W pierwszym etapie działania Hipiska, jego użytkownicy zadali prawie **tysiąc pytań**. Każde z tych pytań zostało zapisane w bazie danych wyszukiwarki. Aby ustrzec się przed złośliwymi atakami, pytania były sprawdzane pod względem długości (zbyt długie i zbyt krótkie pytania były odrzucane), sensowności (np. sprawdzałem czy pytanie nie zawiera zbyt długich ciągów liter) i zawartości (odrzucane były chociażby pytania zawierające nietypowe znaki). Ponadto zebrany materiał był raz jeszcze przeglądany i ewentualne

[6]Open Directory Project, http://www.dmoz.org

[7]Katalog stron internetowych Wirtualna Polska, http://www.wp.pl

[8]Katalog stron internetowych Onet, http://www.onet.pl


Rysunek 2.1: Reklamowe logo serwisu **www.hipisek.pl**

pytania, będące wynikiem zabaw użytkowników, bądź zwyczajnych błędów były usuwane (operacji tej poddałem też pytania zapisane w tym samym okresie i wysłane z tego samego numeru IP komputera). Zastosowanie tak prostych metod wystarczyło by korpus ten nazwać czystym.

Materiał ten stanowi jedyny korpus pytań wyszukiwarki, jaki udało mi się zebrać. Jego główną zaletą jest adekwatność - pytania te zostały bowiem zadane przez użytkowników systemu QA[9]. Co więcej goście portalu wielokrotnie informowani byli o obsługiwaniu tylko pytań o czas i miejsce, dlatego ten rodzaj przykładów pojawia się w źródle najczęściej[10]. Korpus ten stanowi dobrą bazę do rozważań czym jest pytanie o czas i miejsce oraz zastąpieniu nie popartej niczym (poza intuicją) definicji 2.2. Stanowi też dobrą wskazówkę, na jaki rodzaj pytań oczekują odpowiedzi użytkownicy, oraz o co w istocie pytają. Wszystkie te informacje mają kluczowe znaczenie w Rozdziale 3, w którym znajduje się opis i klasyfikacja pytań.

### Podsumowanie zebranego materiału korpusowego

Aby zbudować dobrą bazę lingwistyczną, dla formułowanych w pracy hipotez, oraz zapewnić bogaty materiał testujący, przeszukałem trzy różne źródła korpusowe: artykuły prasowe, strony internetowe oraz pytania zadawane wyszukiwarce. Zebrałem dużą ilość pytań wszelkiego rodzaju, oraz niewielką lecz znaczącą liczbę (ok. 700 różnych) pytań zadawanych wyszukiwarce. Dane dotyczące wielkości korpusów, zawiera tabela 2.1. Do materiału korpusowego będę odwoływał się w trakcie całej pracy.

[9]Użytkownicy zostali poddani, pewnej manipulacji socjotechnicznej, albowiem informacja o tym, że serwis dopiero powstaje nie była nadmiernie eksponowana, natomiast sam system odpowiadał już w dość przewrotny i co najważniejsze zabawny sposób, korzystając z łatwych do zaimplementowania metod wyszukujących typu słowo-klucz. Podobny zabieg, nazywany w literaturze *experiment Wizard of Oz*, przeprowadzony w celu zbadania oczekiwań użytkowników od systemu QA typu *chatterbot* opisano np. w pracy [10].

[10]Był to kolejny zabieg socjotechniczny, mający na celu zwiększenie ilości podawanych pytań o czas i miejsce. W wielu miejscach witryny umieściłem stwierdzenie, że odpowiadanie na inny typ pytań [niż o czas i miejsce] działa znacznie gorzej, co nie było do końca zgodne z prawdą. Użyta metoda przeszukiwania była w pewien sposób skalibrowana na wyrażenia czasowo-przestrzenne lecz nie wpływało to istotnie na jakość odpowiedzi ze względu na typ pytania. Takich *zabiegów* było więcej, jednak ich opis nie jest przedmiotem niniejszej pracy.

Tabela 2.1: Wielkość zebranego materiału korpusowego

| Nazwa źródła | Ilość pytań ogółem | Ilość różnych pytań | Ilość wyrazów |
|---|---|---|---|
| Artykuły prasowe | nie dotyczy | 10375 | 79816 |
| Internet | nie dotyczy | 1004384 | 8689791 |
| Hipisek.pl | 1495 | 717 | 3521 |

## 2.2. Konceptualny model odpowiadania na pytanie

W podrozdziale przedstawiam wizję projektu, zbudowanego jako efekt niniejszej pracy magisterskiej. W paragrafie opisuję elementy z jakich składa się wymyślony przeze mnie system, oraz problemy jakie trzeba było w jego ramach rozwiązać.

### 2.2.1. Baza wiedzy

Oczywistością jest stwierdzenie, że aby odpowiadać na pytanie trzeba posiadać jakąkolwiek wiedzę. Odpowiadanie jest bowiem w pewnym sensie przekazywaniem informacji osobie zadającej pytanie (czy informacja ta, czyli odpowiedź, satysfakcjonuje adresata jest kwestią osobną). W procesie przetwarzania pytania istotne jest więc zdefiniowanie źródła wiedzy, z którego czerpać będziemy odpowiedzi.

Zgodnie z tematem pracy bazą wiedzy jest **tekst niesformatowany** (tzw. *plain-text*) w postaci artykułów prasowych publikowanych w sieci Internet. W przeciwieństwie do innych źródeł danych, często wykorzystywanych w systemach QA (takich jak ontologie np. **WordNet**,[11] gotowe bazy wiedzy np. **DBPedia**[12] lub teksty ustrukturyzowane np. **Wikipedia**[13]), tekst taki jest trudny do przetwarzania w systemach typu QA, m.in. dlatego że zawiera bardzo niewiele **metainformacji**. Przyjmijmy, na potrzeby niniejszej pracy, że:

**Definicja 2.3** ***Metainformacją tekstu źródłowego*** *nazywamy każdą daną związaną z tym tekstem, a nie będącą jego treścią. Metainformacja opisuje dokument, jego budowę, dane które zawiera lub sposób ich reprezentacji.*

W tym rozumieniu **metainformacją** może być np. język artykułu, autor, podział na logiczne części (np. wyróżnienie definicji), data opublikowania. Prosta intuicja, prowadzi do stwierdzenia, że **metainformacje** mogą mieć kolosalny wpływ na proces odpowiadania na pytanie.

Przyjrzyjmy się np. pytaniu:[14]

Kiedy odbędzie się następny mecz na boisku LZS w Zasutowie?

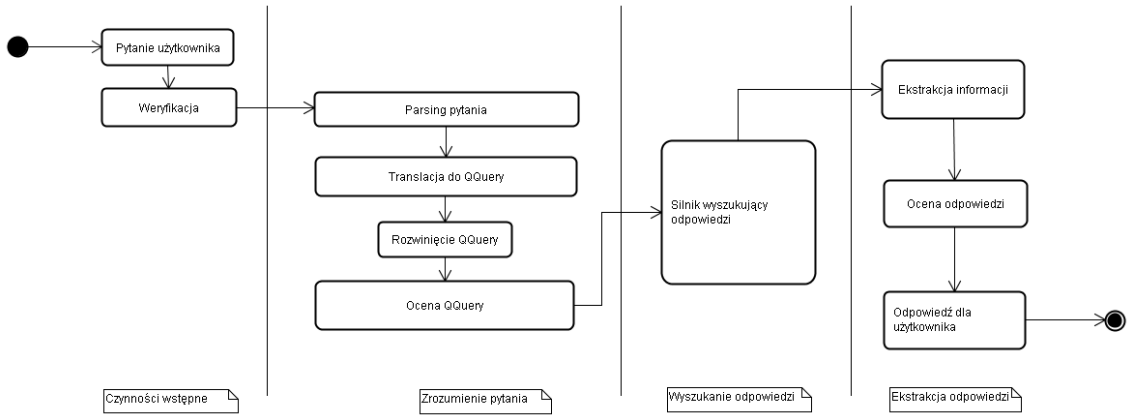
Jeśli system odnajdzie informacje o przyszłych meczach w więcej niż jednym dokumencie (i będą to różne wydarzenia), oraz pojawią się literalnie jako „następny mecz", będzie trzeba rozstrzygnąć, który z nich rzeczywiście oznacza *następny*. Pomóc może tu np. data opublikowania artykułu, dzięki której łatwo sprawdzić, który artykuł jest aktualny, a który

[11] WordNet a lexical database for the English language, http://wordnet.princeton.edu

[12] The DBPedia Knowledge Base, http://dbpedia.org

[13] Wikipedia, the free encyclopedia, http://www.wikipedia.org

[14] Źródło: projekt **Hipisek.pl**




Rysunek 2.2: Proces odpowiadania na pytanie

już przestarzały [15]. System uwzględniający **metainformacje** w swojej bazie wiedzy będzie więc potencjalnie efektywniejszy.

Pierwszym postulatem dla tworzonej bazy wiedzy będzie więc zapewnienie dołączenia do tekstów źródłowych **metainformacji**, które pomogą (a niejednokrotnie pewnie i umożliwią) podanie odpowiedzi na zadane pytanie.

Poza metainformacjami, baza wiedzy z założenia nie będzie elementem złożonym. Wszystkie artykuły, zgodnie z tematem pracy będą przechowywane w postaci tekstu niesformatowanego. Pod względem ich zawartości, w szczególny sposób skupiam się na artykułach prasowych (umieszczanych w ramach internetowych portali), ale założenie to nie jest w pracy kluczowe.

Proces pobierania i indeksowania artykułów oraz tworzenia z nich **bazy wiedzy** przedstawiony jest w Rozdziale 5.

### 2.2.2. Proces przetwarzania pytania

Diagram przetworzenia pytania przedstawiam na rysunku 2.2. Składa się on z czterech etapów, w jakich może znajdować się pytanie od momentu jego zadania po podania przez system odpowiedzi. Proces został podzielony na moduły, które realizują specjalizowane zadania związane z odpowiadaniem.

### 2.2.3. Czynności wstępne

Początek istnienia pytania to oczywiście jego zadanie przez użytkownika. Przyjmuję, że pytanie zostanie wprowadzone do systemu w postaci zdania w języku polskim. Głównym zadaniem tego krótkiego etapu jest czysto techniczna kwestia oczyszczenia pytania z błędów (np. wyeliminowanie zbędnych białych znaków) oraz obrona przed atakami hakerów. Sprawdzone w procesie weryfikacji pytanie zostanie przekazane dalej, po czym rozpocznie się właściwy proces odpowiadania.

[15] Oczywiście nie jest to konieczne, jeśli w artykule pojawią się daty w formie bezwzględnej, czyli chociażby 1 maja 2008 roku. Łatwo jednak zauważyć, że data w takiej postaci pojawia się raczej rzadko.

### 2.2.4. Zrozumienie pytania

Jednym z głównych problemów systemów QA jest działanie na styku *język naturalny - język sztuczny*. Granicę między obydwoma światami przekraczamy w procesie QA co najmniej dwukrotnie: zadając pytanie oraz wyszukując odpowiedź. W pierwszym właściwym etapie przetwarzania pytania zajmę się pierwszym z wymienionych przypadków, czyli problemem translacji zadanego pytania na język zrozumiały przez komputer. Na potrzeby pracy przyjmuję, że:

**Definicja 2.4** ***Zrozumienie pytania*** *to proces przetłumaczenia pytania zadanego w języku naturalnym do zapytania w języku formalnym, które komputer (system QA) będzie w stanie przetwarzać.*

Proces zrozumienia pytania podzieliłem na trzy moduły:

1. **Moduł parsingu** - zajmujący się rozbiorem gramatycznym zdania,
2. **Moduł translacji** - w którym następuje przetworzenie do postaci formalnej QQuery[16],
3. **Moduł rozwinięcia** - w którym rozwiązane zostaną problemy takie jak synonimia pojęć.

W pierwszym z wymienionych modułów następuje analiza językowa wprowadzonego przez użytkownika zdania. Etap ten przetwarza pytanie zapisane w postaci ciągu znaków do ciągu **leksemów**. Przyjmijmy na potrzeby pracy, że:

**Definicja 2.5** ***Leksemem*** *nazywamy abstrakcyjną jednostkę słownikową reprezentującą dany wyraz i posiadającą: formę bazową, formy fleksyjne, synonimy, oznaczenie części mowy i flagi semantyczne (np. imię, nazwa miasta).*

Kolejnym i właściwie najważniejszym modułem jest **moduł translacji**, w którym na podstawie informacji dostarczonych z poprzedniego modułu, zbudowane zostanie pytanie w postaci sformalizowanej. Ponieważ trudno oczekiwać, aby tłumaczenie takie było jednoznaczne, zostanie wprowadzony specjalny moduł oceny trafności translacji, tak aby możliwie szeroko uwzględnić intencje użytkownika zadającego pytanie.

Ostatni z modułów zajmuje się problemem nieścisłości języka naturalnego. Przyjrzyjmy się raz jeszcze poprzedniemu przykładowi:

Kiedy odbędzie się następny mecz na boisku LZS w Zasutowie?

Jeśli nawet uda się zbudować bardzo dobry moduł translacji do postaci formalnej, to w jaki sposób zachowa się wyszukiwarka odpowiedzi napotykając pojęcia takie jak **następny**? Nie możemy przecież wyszukiwać tylko napisów postaci: następny, następna, następnego etc. Potrzebny jest jakiś sposób na rozwinięcie tego zestawu pojęć do chociażby synonimów (np. kolejny, nadchodzący) czy innych relacji (chociażby jeżeli coś odbywa się

[16]Pojęcie **QQuery** zostanie wprowadzone i opisane w Rozdziale 3.

po czymś, to znaczy że jest następne)[17]. W tym celu wprowadziłem do modelu opisywany tutaj moduł, którego zadaniem będzie rozwijanie pojęć pojedynczych do zbiorów pojęć-słów, które w tekście źródłowym mogą reprezentować szukaną informację.

☞ **Uwaga** Przedstawione tu etapy nie mają charakteru arbitralnego, tzn. nie muszą być (i nie będą) realizowane w postaci osobnych modułów programowych. Na przykład ze względów efektywnościowych etap rozwinięcia o synonimy w implementacji systemu **Hipisek.pl** jest realizowany wraz z parsingiem.

Dokładny opis realizacji etapu zrozumienia pytania znajduje się w Rozdziale 4.

### 2.2.5. Wyszukiwanie

Głównym zadaniem etapu wyszukiwania jest określenie fragmentów tekstów, z których nastąpi właściwa ekstrakcja wiedzy. Na podstawie zbudowanego zapytania, system będzie musiał dostarczyć listę fragmentów dokumentów, w których potencjalnie znajdują się istotne informacje, zawierające odpowiedzi na pytanie.

W tym etapie procesu QA po raz kolejny następuje przejście między światem rzeczywistym - czyli tekstami źródłowymi zapisanymi w języku naturalnym, a światem sztucznym - czyli komputerowym. Tym razem nie można jednak liczyć na dokładne metody dziedziny NLP[18], których obliczeniowa złożoność jest zbyt duża. Dlatego na etapie wyszukiwania informacji, skupię się na metodach statystycznych, których wymagania na moc obliczeniową są znacznie mniejsze.

Podstawowym zadaniem etapu wyszukiwania nie jest podanie konkretnych odpowiedzi, a jedynie zawężenie obszaru jej przeszukiwania, dzięki temu w kolejnym etapie, będę mógł skorzystać z metod bardziej wyrafinowanych. Jest to więc pewnego rodzaju próba rozwiązania problemu złożoności zadania wyszukania konkretnej informacji. System nie będzie bowiem w sposób ślepy przeszukiwał wszystkich dostępnych mu tekstów źródłowych, ale stopniowo ograniczał przestrzeń przeszukiwania, by docelowo (w ramach opisywanego etapu) otrzymać tekst na tyle krótki, by zastosować na nim metody wymagające więcej mocy komputera. Dla rzeczonego przykładu:

> Kiedy odbędzie się następny mecz na boisku LZS w Zasutowie?

System powinien wykonać co najmniej wymienione czynności:

- Odrzucić wszystkie teksty niezwiązane w żaden sposób z miejscowością Zasutowo,
- Wyeliminować teksty nie opisujące spotkań na boisku,
- Usunąć fragmenty i zdania w których brak jest informacji o meczach i czasie ich trwania.

Problematyczne jest tu arbitralne wyznaczenie końca procesu „odcinania" kolejnych fragmentów. Należy znaleźć tu *złoty środek* między dwoma ekstremalnymi podejściami:

[17] Problem taki można rozwiązać wprowadzając mechanizmy wnioskowania do systemu QA. W rozpatrywanym przykładzie można by próbować przeprowadzić wnioskowanie w oparciu o listę znalezionych spotkań na rzeczonym boisku. Wnioskowanie nie jest jednak przedmiotem tej pracy. Więcej o teorii wnioskowania czasowo-przestrzennego można znaleźć w źródle [11].

[18] NLP – Natural Language Processing.

- **Usuwanie skąpe** – polegające na usuwaniu minimalnych partii tekstu (pozostawiające dużo tekstów redundantnych i zbędnych),
- **Usuwanie zachłanne** – polegające na usuwaniu maksymalnie dużych partii tekstu (także tych, które mogą zawierać potrzebne nam informacje, ale z małym prawdopodobieństwem).

Zbyt skąpe usuwanie może spowodować przeładowanie znalezionego materiału (będzie go po prostu zbyt dużo), natomiast jego nadmierna zachłanność może spowodować otrzymanie bardzo skąpego tekstu, nie mającego większej wartości.

Oczywiście intuicja oraz wstępne eksperymenty związane z pracami nad zbieraniem korpusu w ramach serwisu **Hipisek.pl** podpowiadają, że część odpowiedzi (szczególnie tych literalnie zapisanych w tekście źródłowym), zostanie znaleziona już na tym etapie. System wyszukiwania musi uwzględnić takie sytuacje i nadać im odpowiednią wagę.

W ramach etapu wyszukiwania należy wprowadzić odpowiedni sposób oceny wyszukanych fragmentów, tak aby teksty o najwyższej jakości były w kolejnym etapie w pewien sposób faworyzowane. Ocena taka powinna brać pod uwagę:

- Jakość tekstu,
- Trafność wyszukiwania (dopasowanie słów kluczowych),
- Jakość metody wyszukiwania (w procesie wyszukiwania możemy korzystać z różnych metod, których jakość będzie się różnić).

Wynikiem etapu wyszukiwania będą uporządkowane wg oceny fragmenty tekstu, wraz ze źródłem z którego pochodzą. Wraz z przetłumaczonym pytaniem trafią one do ostatniego etapu procesu QA zajmującego się konkretyzowaniem odpowiedzi.

Proces wyszukiwania odpowiedzi opisano dokładnie w Rozdziale 5.

### 2.2.6. Ekstrakcja odpowiedzi

Kiedy pytanie zostanie zrozumiane przez komputer, czyli otrzymamy jego reprezentację formalną, oraz gdy wyszukamy fragmenty tekstów źródłowych, potencjalnie zawierające odpowiedź, system QA może przystąpić do ostatniego etapu przetwarzania, w którym następuje sformułowanie konkretnej odpowiedzi. Przyjmiemy na potrzeby pracy, że podając odpowiedź nie interesuje nas jej forma, dlatego odpowiedzią na pytanie w ramach rozpatrywanego przykładu:

> Kiedy odbędzie się następny mecz na boisku LZS w Zasutowie?

Jest zarówno zdanie:

> Następny mecz na boisku LZS w Zasutowie odbędzie się w piątek.

Jak również formularz, taki jak w tabeli 2.2.

Głównym problemem z jakim należy się zmierzyć w etapie ekstrakcji odpowiedzi jest ekstrakcja (znalezienie) odpowiedzi z dostarczonych przez wyszukiwarkę fragmentów tekstu. Zakładam na tym etapie, że znalezione fragmenty wystarczają do udzielenia odpowiedzi na zadane pytanie. Ekstrakcja wiedzy będzie następować w oparciu o metody regułowe oraz metody statystyczne, których podstawowym założeniem jest bliskość pojęć (w sensie ilości dzielących wyrazy znaków) i ich opisów.

Tabela 2.2: Lista meczów na boisku LZS w Zasutowie (lista fikcyjna)

| | |
|---|---|
| Polonia Bytom vs. Zagłębie Lublin | 01.03.2008r. |
| MKS Osadnik vs. Pogoń Barlinek | 01.04.2008r. |
| Sokół Szamocin vs. Lech Poznań | 21.04.2008r. |
| Odra Wodzisław vs. Arka Gdynia | 27.05.2008r. |

Kolejny z problemów tego etapu polega na występowaniu częstych informacji sprzecznych w bazie artykułów źródłowych. Chodzi mi w tym miejscu o takie fragmenty tekstu jak:

- Stwierdzenia i zdania oznajmująca będące w rzeczywistości subiektywnymi opiniami i wyrazem przekonań autora
- Domniemania i tzw. *„gdybanie"* na temat przyszłości
- Domysły nadawcy fragmentu tekstu
- Plany i zapowiedzi (np. działań rządu, wyjazdów)

Aby poradzić sobie z tym problemem system QA będzie musiał wprowadzić system oceny wiarygodności, każdej ze znalezionych informacji, po czym w ostatecznym procesie formułowania odpowiedzi brać pod uwagę przede wszystkim wiedzę o wiarygodności najwyższej, stopniowo obniżając próg akceptacji. Co więcej warto jest identyfikować fakty całkowicie sprzeczne, np. postaci *X odbędzie się w Y* i *X nie odbędzie się w Y*, tak aby system dobrze wypełniał podstawowy postulat otwierający niniejszy rozdział[19].

Mając tak przygotowane pojęcia, system QA powinien starać się sformułować odpowiedzi na zadane pytanie. Liczba mnoga nie jest tu użyta przypadkowo, gdyż może się zdarzyć, że na dane pytanie można odpowiedzieć na wiele możliwych sposobów (i tak rzeczywiście często jest). System powinien umieć odpowiednio ocenić wagę każdej ze sformułowanej odpowiedzi, oraz podać je w kolejności malejącej wg ocen. Ocena każdej z odpowiedzi powinna brać pod uwagę:

- Oceny cząstkowe użytych informacji,
- Stopień wypełnienia pytania (czyli ilość informacji przedstawionych w stosunku do spodziewanych),
- Jakość źródła.

Tak wygenerowane odpowiedzi system QA poda użytkownikowi.
Realizacja etapu ekstrakcji odpowiedzi została opisana w Rozdziale 6.

### 2.2.7. Podsumowanie modelu

Przyjęty model odpowiadania na pytanie, choć i tak mocno uproszczony, jest dość złożony. Model został stworzony w oparciu o moduły realizujące poszczególne pomysły przetwarzania QA. Modułowa budowa umożliwia jego potencjalną rozbudowę w przyszłości, poprzez dodawanie kolejnych elementów.

[19]Problemy wymienione w tym paragrafie są zadaniem trudnym. W pracy ograniczyłem się do ocen statystycznych, które nie analizują znalezionych odpowiedzi w tak dokładny sposób.

W opisie teoretycznym nie brałem pod uwagę kwestii technicznych i różnego rodzaju ograniczeń związanych ze złożonością, choć miałem je ciągle na uwadze. Kwestie techniczne przybliżone zostaną w kolejnych rozdziałach.

ROZDZIAŁ 3

# Formalna reprezentacja pytania za pomocą QQuery

W rozdziale przedstawiam proponowaną na potrzeby pracy formalizację pojęć dotyczących pytań i ich klasyfikację, które oparłem o zebrany materiał korpusowy. Następnie wprowadzam formalizm reprezentowania pytań za pomocą QQuery oraz dokładnie omawiam jego elementy.

## 3.1. Sprecyzowanie pojęć dotyczących pytań o czas i miejsce

W oparciu o zebrany materiał korpusowy, którego procedurę pozyskania przedstawiłem w Rozdziale 2 dokonałem formalizacji pojęć dotyczących: pytań ich typów oraz odpowiedzi. Pracę wykonałem dokonując obserwacji zebranych pytań, zliczając statystyki i ekstrahując odpowiednie podkorpusy.

### 3.1.1. Analiza zebranego materiału korpusowego

Pracę nad uściśleniem kluczowych pojęć rozpocząłem od ręcznego przeglądania zebranych źródeł korpusowych **korpusu pytań wyszukiwarki** (w skrócie korpusu wyszukiwarki **KPW**), fragmentów **ogólnego korpusu pytań stron internetowych** (w skrócie korpusu internetowego **KPI**) i **ogólnego korpusu pytań artykułów prasowych** (w skrócie korpusu artykułów **KPA**)[1]. W przypadku źródeł **KPI** oraz **KPA** selekcji fragmentów poddanych obserwacji dokonywałem poprzez:

- Wybieranie losowych próbek pytań,
- Wybieranie pytań zaczynających się od podanych fraz.

Pierwszym z badań, jakich dokonałem na zebranym materiale, było zbadanie rozkładu wystąpień pytań w zależności od pierwszego wyrazu pytania. Statystyki każdego z korpusów, ograniczone do dziesięciu najczęstszych wystąpień, znajdują się w tabeli 3.1.

Obserwując statystyki, łatwo zauważyć, że źródła **KPA** i **KPI** są do siebie dość podobne (cztery najpopularniejsze wyrazy rozpoczynające pytania są identyczne w obu źródłach, znajdują się tylko na innych pozycjach). Widać też dominującą rolę pytań o miejsce (pierwsza pozycja pytań zaczynających się od **gdzie**) i czas (pozycja trzecia pytań zaczynających się od **kiedy**) w źródle **KPW**. Dodatkowo w źródłach **KPA** i **KPI** znaczną rolę odgrywają pytania, rzadko rozpatrywane przez wyszukiwarki QA (np. pytania zaczynające się od „**a**": *A jak wyglądają krajobrazy Wisły?* lub „**może**": *Może coś wymyka się naukowym opisom?*)[2].

Do podobnych wniosków prowadzą obserwacje konkretnych przykładów pytań z zebranych źródeł. Po przejrzeniu fragmentów (losowe próbki po 1000 pytań) źródeł **KPA** i **KPI** i całości źródła **KPW** zaobserwowałem następujące własności korpusów:

- Źródła **KPA** i **KPI** są pod względem rozpatrywanej dziedziny QA, bardzo podobne, jednakże pierwsze z nich zawiera pytania o znacznie wyższej *jakości*, tzn. z mniejszą

[1] Procedura pozyskania korpusów, oraz charakterystyka ich źródeł znajduje się w Rozdziale 2.
[2] Podobny problem opisano w pracy [12].

Tabela 3.1: Występowanie pytań zaczynających się od podanego słowa w zebranym materiale korpusowym

| KPA | | KPI | | KPW | |
|---|---|---|---|---|---|
| czy | 1670 | a | 146804 | gdzie | 153 |
| co | 1519 | czy | 121555 | czy | 148 |
| jak | 1160 | jak | 54623 | kiedy | 129 |
| a | 772 | co | 48487 | kto | 55 |
| dlaczego | 504 | może | 33420 | co | 31 |
| nie | 257 | nie | 32497 | ile | 23 |
| może | 237 | to | 28918 | dlaczego | 21 |
| ale | 193 | i | 28715 | kim | 20 |
| po | 185 | ale | 18018 | podaj | 13 |
| kto | 162 | no | 17046 | jak | 13 |
| wszystkich | 10375 | wszystkich | 1004384 | wszystkich | 717 |

ilością błędów typograficznych, gramatycznych i stylistycznych, natomiast tematyka pytań źródła **KPI** jest znacznie szersza,

- Pytania źródła **KPA** są w dużej mierze nieprzydatne dla badań nad dziedziną QA, np. mają charakter *wywiadu* (np. *Czy Beckett tłumaczył Panu, o co chodzi w Krappie?*), dotyczą poglądu danej osoby (np. *Jak może wyglądać lekcja polskiego z jego użyciem?*, *A co Panią drażni?*), mają charakter retoryczny (np. *Może to własna ludzka słabość nas do tego ciągnie?*), jednakże pewna klasa pytań przydatnych może być łatwo wyekstrahowana poprzez selekcję pytań zaczynających się od podanych fraz (np. *Gdzie są...?*),

- Pytania źródła **KPI** są również w dużej mierze nieprzydatne dla badań w niniejszej pracy, np. mają charakter kontekstowy (np. *Jak mogę się pozbyć tego błędu?*), dotyczą opinii adresata zbiorowego (np. *Myślicie że to jakiś Obcy?*), lub podtrzymują rozmowę (szczególnie pytania zaczynające się od „**a**" np. *A może jest inny powód?*). Podobnie jednak jak w źródle **KPA** podkorpus pytań przydatnych dla celów pracy, można w łatwy sposób uzyskać automatycznie,

- Odsetek pytań o czas i miejsce w źródłach **KPA** i **KPI** jest wyjątkowo niski,

- Źródło **KPW**, choć objętościowo małe zawiera bogaty i adekwatny do rozpatrywanej dziedziny zbiór pytań, niestety ograniczony tematycznie do polityki, sportu oraz plotek ze świata *showbiznesu* (spowodowane jest to indeksowaniem przez serwis **Hipisek.pl** dwóch portali prasowych i jednego plotkarskiego: www.tvn24.pl, www.onet.pl[3] oraz www.pudelek.pl),

- Problemowym zjawiskiem występującym w źródle **KPW** jest pewna ilość pytań typu życzeniowego, dotyczących przewidywania przyszłości (np. *Kiedy będzie koniec kryzysu?*) i mających zabarwienie humorystyczne (np. *Gdzie w Poznaniu można kupić świeże mięso?*). Takie pytania nie mają istotnego znaczenia w budowanym modelu systemu QA, ale mają pewne znaczenie praktyczne, pokazując, o co może zapytać przeciętny użytkownik gotowej wyszukiwarki.

[3]W docelowej implementacji zrezygnowałem z indeksowania artykułów www.onet.pl, jednak zbierając materiał korpusowy funkcjonalność taka była ciągle włączona.

Powyższe obserwacje skłoniły mnie do zbudowania **podkorpusu pytań o czas** i **podkorpusu pytań o miejsce** w oparciu o prostą selekcję fraz rozpoczynających pytanie (podobnie jak przy zbieraniu materiału korpusowego, patrz Rozdział 2). Na potrzeby pracy przyjmijmy, że:

**Definicja 3.1** ***Podkorpusem*** *nazywamy dowolny podzbiór tekstu danego korpusu, wybrany ze względu na badaną właściwość.*

O klasyfikacji pytania ze źródeł **KPA** i **KPI** do podkorpusu pytań o czas oraz podkorpusu pytań o miejsce, decydowała fraza rozpoczynająca.

Wybierając taki sposób ekstrakcji podkorpusów milcząco założyłem, że o typie pytania decyduje jego początek, co nie zawsze jest prawdą (zależy to od przyjętej taksonomii pytań, patrz źródło [13]). Podobne założenie przyjmują często systemy oparte o szablony pytań (tzw. *question templates*) np. [14]. Problem ten rozpatrzony zostanie w podrozdziale 3.2.2. Warto także nadmienić, że statystyki korpusu **KPW** zawarte w tabeli 3.1 w pewien sposób uprawdopodobniają hipotezę, że *język zapytań kierowanych przez użytkowników jest ubogi i składa się ze zdań rozpoczynających się od fraz należących do niewielkiego zbioru fraz rozpoczynających pytanie*, przez co budowanie rozbudowanych gramatyk analizy pytania, może okazać się zadaniem przynoszącym niewielki zysk w sensowności działania gotowego systemu.

Zbudowane podkorpusy zawierają **2148** pytań o czas oraz **7018** pytań o miejsce. Stanowią wraz ze źródłem **KPW** bazę do definicji i stwierdzeń całego niniejszego rozdziału (źródła te dla uproszczenia będę nazywał dalej **korpusem bazowym** - mówiąc o obu źródłach bądź **korpusem pytań o czas** lub **korpusem pytań o miejsce** - mówiąc o odpowiednich podkorpusach). Pięć losowych pytań z każdego z podkorpusów znajduje się w tabeli 3.2 (zachowano oryginalną pisownię).

Tabela 3.2: Przykładowe pytania z korpusu pytań o czas i miejsce

| **Pytania o czas** |
|---|
| Od kiedy Millenium ma darmowe wypłaty z Euronetów ? kiedy się znajdą? |
| Kiedy był bymieniany olej w szkrzyni? |
| Od kiedy mniej więcej zaczęto montować USB na płytach? |
| Od kiedy na Euro grają kluby? |

| **Pytania o miejsce** |
|---|
| gdzie w Łodzi jest Dolina Szwajcarska ? |
| Gdzie został pochowany Bronisław Geremek? |
| Gdzie produkowane są rewery wheeler? |
| Gdzie jest księgarnia z książkami technicznymi? |
| W którym kraju jest najniższe bezrobocie? |

### 3.1.2. Formalizacja pojęć: pytania o czas i pytania o miejsce

Na podstawie przykładów znajdujących się w **korpusie bazowym** wprowadźmy na potrzeby pracy następującą definicję pytania:

**Definicja 3.2** ***Pytanie*** *jest zdaniem języka naturalnego, którego głównym celem jest przekazanie polecenia o udostępnieniu nadawcy pytania pewnej informacji bądź ich zbioru.*

Pytania ze względu na ilość oczekiwanych informacji możemy podzielić na **pytania proste** i **pytania złożone**. Zdefiniujmy:

**Definicja 3.3** ***Pytanie proste*** *to pytanie, które wymaga od adresata odnalezienia tylko jednej informacji lub zbioru informacji tego samego typu, na jeden zadany temat.*

**Definicja 3.4** ***Pytanie złożone*** *to pytanie, które wymaga od adresata odnalezienia dwóch lub więcej informacji na jeden lub kilka z zadanych tematów.*

W rozumieniu powyższych definicji **pytaniem prostym** jest np. *Która jest godzina?* lub *Wymień terminy zakończenia kolejnych etapów* natomiast **pytaniem złożonym** jest np. *Podaj datę i miejsce najbliższego szczytu Unii Europejskiej*. Czasami klasyfikacja pytań na proste i złożone jest zadaniem dyskusyjnym np. pytanie *Kiedy urodziła się córka prezydenta Polski?* może być traktowane jako pytanie złożone, składające się z pytań[4]:

1. Kto jest prezydentem Polski?
2. Kto jest córką osoby będącej odpowiedzią na pytanie (1)?
3. Kiedy urodziła się osoba będąca odpowiedzią na pytanie (2)?

☞ **Uwaga** W toku dalszej pracy zajmę się jedynie **pytaniami prostymi**. Pytania złożone są osobnym problemem w systemach QA, wymagającym często specyficznych metod przetwarzania[5]

Ze względu na charakter pojęć występujących w pytaniu możemy wprowadzić podział na pytania **kontekstowe** i **bezkontekstowe**[6]. Zdefiniujemy:

**Definicja 3.5** ***Pytanie bezkontekstowe*** *to pytanie, zawierające wszystkie niezbędne informacje, wymagane do określenia typu informacji, której żąda nadawca.*

**Definicja 3.6** ***Pytanie kontekstowe*** *to pytanie, którego zadanie nie umożliwia określenia w pełni typu informacji, której wymaga nadawca. Dopiero osadzenie pytania w kontekście całej konwersacji (wypowiedzi przeszłych i przyszłych) umożliwia dokładne określenie typu informacji, będącej odpowiedzią na zadane pytanie.*

[4]Por. dekompozycja pytań złożonych w [15].

[5]Czasem jednak wystarczy proste założenie, że pytania złożone zawsze dają się rozłożyć na sumę pytań prostych np. pytanie *Podaj datę i miejsce szczytu Unii Europejskiej*.

[6]Pytania bezkontekstowe są domeną wyszukiwarek QA, które całkowicie pomijają pytania kontekstowe (por. np. usuwanie kontekstowości z pytań korpusu typu FAQ w [4]. Pytania kontekstowe rozpatrywane są w przypadku budowania tzw. *QA chatbotów* np. [16] lub [10].

Pytaniem bezkontekstowym w rozumieniu powyższej definicji będzie np. pytanie *Kiedy urodziła się córka prezydenta Polski?*. Natomiast pytaniem kontekstowym będzie pytanie zadane przez osobę B w poniższym dialogu:

```
A: Jesteśmy teraz w Warszawie.
B: A ile mieszkańców ma to miasto?
```

Kontekstowość tego pytania polega na tym, że rozpatrując je samodzielnie nie możemy dokładnie określić, o jakie konkretnie miasto chodzi, natomiast umożliwia to osadzenie pytania **w kontekście** całej wypowiedzi.

W pracy zajmuję się wyszukiwarką QA, w której niemożliwe jest prowadzenie dialogu, dlatego zajmę się jedynie **pytaniami bezkontekstowymi**.

Ze względu na typ informacji, której poszukuje nadawca pytania, możemy zdefiniować dwa podstawowe pojęcia pracy:

**Definicja 3.7** ***Pytanie o czas*** *jest to pytanie, którego odpowiedzią jest dowolne pojęcie wyrażające czas, a temat pytania jest bezpośrednio powiązany z wykonaniem pewnej akcji, spełnienia określonego warunku lub znalezienia się w określonym stanie w odpowiadającemu odpowiedzi momentowi czasowemu.*

Pierwszy z członów powyższej definicji ogranicza pytania o czas do pytań, których odpowiedzią jest pewne pojęcie wyrażające czas. Drugi z nich zapewnia powiązanie tematu pytania (czyli pojęcia, o które pyta nadawca) z tym pojęciem, dzięki czemu pytaniem o czas nie będzie na przykład zdanie: *Podaj daty zapisane na marginesie*. Definicja ta bardzo zawęża dziedzinę pytań o czas. W literaturze można znaleźć bardziej ogólne definicje. Na przykład w źródle [17], przez **pytania o czas** rozumie się pytania, które nie tylko wymagają jako odpowiedzi pewnego *wyrażenia czasowego*, ale również pytania, które dotyczą pewnej relacji czasowej (np. *co wydarzyło się po czymś*) oraz pytania dotyczące pewnego przedziału czasowego. Jednak takie rodzaje pytań nie zostaną omówione w ramach tej pracy magisterskiej.

**Definicja 3.8** ***Pytanie o miejsce*** *jest to pytanie, którego odpowiedzią jest dowolne pojęcie wyrażające miejsce w przestrzeni (miasto, adres, kraj etc.), a temat pytania jest bezpośrednio powiązany z wykonaniem pewnej akcji, spełnienia określonego warunku lub znalezienia się w określonym stanie, w odpowiadającemu odpowiedzi miejscu w przestrzeni.*

Podobnie jak w przypadku definicji **pytania o czas** w pracy ograniczę się jedynie do pytań ukierunkowanych na odpowiedź będącą wyrażeniem przestrzennym. Pominę problemy takie jak *relacje przestrzenne* i pojęcia związane semantycznie z miejscami.

### 3.1.3. Odpowiedź na pytanie

W źródle [13] Jefrrey Pomerantz stwierdza, że:

> Celem zadania QA jest pozyskanie małych porcji tekstu, które zawierają faktyczną odpowiedź na pytanie, w przeciwieństwie do listy dokumentów, tradycyjnie zwracanych przez systemy wyszukiwania informacji.[7]

Z kolei w pracy [18] zawarta jest następująca konkluzja:

> Pomimo że *systemy question answering* są rozwijane w stronę dostarczenia tylko dokładnych odpowiedzi, nasze badania pokazały, że użytkownicy faktycznie preferują fragmenty tekstu na poziomie akapitu (z odpowiednim wyszczególnieniem odpowiedzi).[8]

W pracy przyjmę jednak strategię pierwszą, przesuwając interakcję z użytkownikiem na dalszy plan. Przyjmijmy na potrzeby pracy, że:

**Definicja 3.9** ***Odpowiedź*** *na zadane pytanie to możliwie krótka partia tekstu, która zawiera informację, o którą pyta nadawca pytania.*

W rozumieniu powyższej definicji bardziej adekwatną odpowiedzią na pytanie *Która godzina?* jest tekst postaci: *14:56*, niż zdanie *Aktualnie jest godzina czternasta pięćdziesiąt sześć*. Osobnym problemem jest stopień szczegółowości odpowiedzi, np. problem czy w podanym przykładzie należy podać czas z dokładnością do minuty, sekundy czy jeszcze dokładniej. Jednak ten oraz inne podobne problemy związane z procesem kognitywnym rozumienia odpowiedzi przez człowieka (użytkownika systemu QA) nie zostaną omówione w ramach tej pracy.

## 3.2. Zapytania QQuery

Korzystając z definicji zbudowanych w pierwszej części rozdziału przedstawię sposób reprezentacji pytania w sformalizowany sposób, czytelny dla komputera. Pytanie w tej postaci będzie przetwarzane w trakcie dalszych etapów ekstrakcji informacji i pozyskiwania odpowiedzi.

### 3.2.1. Definicja zapytania QQuery

Przypomnijmy, że zgodnie z definicją 2.4 **zrozumienie pytania** to proces polegający na przetłumaczeniu zdania zapisanego w języku naturalnym, niezrozumiałym dla komputera, na pytanie zapisane w sformalizowany sposób za pomocą języka sztucznego. Przyjąłem, że pytanie reprezentowane będzie za pomocą piątki, którą nazwałem **QQuery**[9].

[7] Jeffrey Pomerantz *A Linguistic Analysis of Question Taxonomies* [13], tłumaczenie własne.

[8] Jimmy Lin, Dennis Quan, Vineet Sinha, Karun Bakshi, David Huynh, Boris Katz, and David R. Karger *What Makes a Good Answer? The Role of Context in Question Answering* [18], tłumaczenie własne.

[9] Termin **QQuery** wprowadzam inspirując się pracą [19], w której występuje podobne, choć prostsze, pojęcie **QTarget**.

**Definicja 3.10** *Niech dane będą: zbiór $L$ dowolnych* ***leksemów*** *języka polskiego (wraz z wyróżnionym elementem pustym $\varepsilon$), zbiór $L_c$ będący zbiorem dowolnych ciągów składających się z elementów zbioru $L$, zbiór $N$ dowolnych napisów języka polskiego, element $t \in T$ nazywany* ***typem pytania****, gdzie $T$ jest zdefiniowanym, zbiorem typów, zwanym* ***taksonomią pytań****, ciąg $s \in L_c$ nazywany* ***tematem pytania****, ciąg $a \in L_c$ nazywany* ***akcją pytania****, zbiór $C \subset L$ będący* ***zbiorem ograniczeń*** *oraz zbiór $P \subset N$ będący* ***zbiorem fraz wyszukujących****. Wtedy piątkę postaci $Q = (t, s, a, C, P)$ nazywamy* ***zapytaniem QQuery****.*

W dalszej części podrozdziału skupię się na wyjaśnieniu pojęć występujących w powyższej definicji. Dokładna specyfikacja implementacyjna pojęcia QQuery zaprogramowana i używana w systemie **Hipisek.pl** znajduje się w dodatku A niniejszej pracy.

### 3.2.2. Typy QQuery - taksonomia pytań

W źródle [13] Jeffrey Pomerantz wymienia następujące typy taksonomii pytań, spotykane w literaturze dotyczącej zadaniom QA i ekstrakcji informacji:

1. Wyrazy Wh- (ang. *Wh- words*)
2. Temat pytania
3. Funkcje oczekiwanych odpowiedzi na pytania
4. Formy oczekiwanych odpowiedzi na pytania
5. Typy źródeł, z których mają zostać pobrane odpowiedzi[10]

W pracy tej nadmienia, że najczęściej stosuje się pierwszy z wymienionych sposobów klasyfikacji, gdyż jest najbardziej intuicyjny i najprostszy. Dodaje również, że często stosuje się łączenie różnych taksonomii w ramach jednego systemu, gdyż:

> ...budując system, nie jest istotne przestrzeganie rygorystyczne teorii lecz najważniejsze jest zbudowanie systemu, który działa.[11]

Dlatego w pracy niniejszej łączę cztery pierwsze z wymienionych taksonomii, aby maksymalnie wykorzystać zalety i możliwości każdej z nich. Ze względu na monotoniczność źródłowej bazy wiedzy, piąta taksonomia oparta o typ źródła, nie znajdzie tu zastosowania.

W wyniku przeprowadzonych obserwacji na **korpusie bazowym**, *taksonomię pytań* opieram na następujących własnościach pytania i odpowiedzi:

1. Typ zwracanej wartości jako odpowiedź (czas lub miejsce),
2. Charakter poszukiwanej wiedzy (wiedza dotycząca przeszłości, aktualny stan, wiedza przewidywana).

[10] Jeffrey Pomerantz *A Linguistic Analysis of Question Taxonomies* [13], tłumaczenie własne.
[11] *ibidem*, tłumaczenie własne.

Pierwszym naturalnym podziałem, jaki narzuca samo sformułowanie tematu pracy, jest podział typu pytania na: **pytania o czas** i **pytanie o miejsce**. Pierwszym z atrybutów typu pytania będzie więc informacja, którym z nich jest analizowane pytanie. Jest to podział determinujący typ odpowiedzi, jakiej poszukiwać ma system.

Drugim z elementów przyjętej typizacji pytań jest charakter wymaganej wiedzy. Obserwując korpus **KPW**, zauważyłem, że użytkownicy wykazują tendencję do:

- Zadawania pytań dotyczących wydarzeń przeszłych, np. *Gdzie była Jola Rutowicz?*
- Zadawania pytań dotyczących bieżących wydarzeń, np. *Gdzie jest Lech Kaczyński?*
- Zadawania pytań *życzeniowych* dotyczących albo planowanych wydarzeń lub spekulacji na temat przyszłości, np. *Kiedy będzie finał Mam Talent?* lub *Kiedy będzie padać?*

Dlatego w przyjętej taksonomii kolejnym wyróżnionym elementem jest jedna z trzech własności:

- PAST, pytania o wydarzenia z przeszłości
- PRESENT, pytania o aktualny stan rzeczy
- FUTURE, pytania dotyczące przyszłości

☞ **Uwaga** Dodatkową wartością typu pytania jest sztuczna wartość **UNKNOWN**, oznaczająca, że nie wiemy jaki jest to konkretnie typ.

Typ zapytania jest pierwszym z elementów QQuery, który determinuje rodzaj szukanej odpowiedzi oraz w pewien sposób ma wpływ na wyszukiwanie konkretnych partii tekstów, z których nastąpi ekstrakcja wiedzy. Reasumując, możemy zdefiniować typ jako parę $t = (t_t, t_s)$, gdzie $t_t$ jest typem odpowiedzi natomiast $t_s$ charakterem wymaganej wiedzy. Natomiast taksonomię $T$ możemy zdefiniować jako zbiór par postaci:

$$T = \{(t_t, t_s) : t_t \in T_t, t_s \in T_s\}$$

Natomiast zbiory $T_t$ oraz $T_s$ definiujemy jako:

$$T_t = \{TIME, PLACE, UNKNOWN\}$$

$$T_s = \{PAST, PRESENT, FUTURE, UNKNOWN\}$$

### 3.2.3. Temat QQuery

Temat pytania możemy w pewien sposób utożsamić z *podmiotem zdania*. Temat w rozumieniu niniejszej pracy, będzie podstawowym pojęciem, na temat którego poszukujemy odpowiedzi. Ponieważ w pracy ograniczyłem się do **pytań prostych**, temat ten będzie pojedynczą *jednostką semantyczną* (co nie znaczy, że pojedynczym leksemem), którą potraktuję jako **główne słowo kluczowe**, w procesie ekstrakcji informacji.

**Definicja 3.11** ***Temat pytania*** *to pojęcie (lub zbiór pojęć) będące podmiotem pytania, o którym informacji szukamy w celu udzielenia odpowiedzi na pytanie.*

Przykładowe tematy pytań, **oznaczone ręcznie** znajdują się w tabeli 3.3.

Tabela 3.3: Ręczne oznaczenie tematu pytania

| Pytanie | Temat |
|---|---|
| Gdzie jest Lech Kaczyński? | Lech Kaczyński |
| Kiedy w Poznaniu pojawią się nowoczesne tramwaje Combino? | tramwaje |
| Gdzie jest Pojezierze Łęczycko-Włodawskie ? | Pojezierze Łęczycko-Włodawskie |
| Gdzie we Wrocławiu są sklepy z rowerami Merida? | sklepy |

### 3.2.4. Akcja QQuery

W większości zadawanych pytań z **korpusu bazowego**, odpowiedź związana jest z wykonaniem pewnej czynności przez *temat pytania*. Na przykład pytanie: *Gdzie poleciał Lech Kaczyński?* można rozumieć jako polecenie podania miejsca, do którego Lech Kaczyński *poleciał*. Kierując się tą intuicją, zdefiniujemy **akcję** jako podstawową czynność, którą wykonuje **temat pytania**.

**Definicja 3.12** ***Akcja pytania*** *to czynność (czynności) lub stan (stany), które wykonuje lub którym podlega* ***temat pytania****.*

Akcja ta nie musi wystąpić literalnie w pytaniu, co wynika z własności języka polskiego (np. w pytaniu *O której koniec balu?* zamiast *O której* ***będzie*** *koniec balu?*). Przykładowe akcje pytań, oznaczone ręcznie znajdują się w tabeli 3.4.

Tabela 3.4: Ręczne oznaczenie akcji pytania

| Pytanie | Akcja |
|---|---|
| Kiedy zacznie się odbudowa? | zacznie się |
| Kiedy ukaże się pełna wersja Tactical Ops? | ukaże się |
| Kiedy w Poznaniu pojawią się nowoczesne tramwaje Combino? | pojawią się |
| Gdzie jest Pojezierze Łęczycko-Włodawskie ? | jest |
| Gdzie we Wrocławiu są sklepy z rowerami Merida? | są |

Akcje pytań posłużą do sformułowania **fraz wyszukujących** oraz pomogą w oznaczeniu relacji semantycznych, podczas procesu odpowiadania na pytanie.

### 3.2.5. Zbiór ograniczeń QQuery

Trzy wcześniej omówione pojęcia umożliwiają reprezentowanie całkiem dużej ilości pytań, jednakże do przedstawienia trochę bardziej złożonych pytań, dodam mechanizm nazwany **zbiorem ograniczeń** (ang. **constraints**). Przez **ograniczenie** (ang. **constraint**) rozumiem tu pewien dodatkowy warunek, jaki musi spełnić **temat pytania** lub **akcja pytania**, aby fragment tekstu będący potencjalną odpowiedzią został zaakceptowany. Ograniczenie to będzie miało postać **zbioru słów kluczowych**.

Kilka systemów QA z dużym sukcesem korzysta z metody wyszukiwania w oparciu o słowa kluczowe (np. [20]). W mojej pracy wykorzystam słowa kluczowe jako warunki, które powinien spełnić tekst aby być dobrą odpowiedzią na pytanie.

Ograniczenia są listą par: **leksem** oraz jego **waga**. Praktycznie do listy ograniczeń trafią wszystkie **słowa znaczące**[12] występujące w pytaniu, które nie zostały użyte w **akcji pytania** bądź **temacie pytania**.

### 3.2.6. Zbiór fraz QQuery

Zbiór fraz zapytania nie będzie pozyskiwany bezpośrednio z pytania lecz wygenerowany przez **reguły translacji**[13]. Frazy są rozszerzeniem słów kluczowych i działają w podobny sposób. Frazy są zbiorem ciągów wyszukujących (w szczególności może być to zbiór pusty), które w pewien sposób pomagają zidentyfikować dokumenty zawierające odpowiedź na pytanie. Koncepcyjnie nie różnią się od słów kluczowych (słowa kluczowe można traktować jako frazy długości jeden), jednak różnią się kluczowo w praktyce. Frazy są dokładniejsze (chociażby przez długość ciągu wyszukującego), przez co statystycznie powinny dawać lepsze wyniki.

Dokładny sposób wykorzystania ograniczeń i fraz w procesie ekstrakcji informacji zostanie przedstawiony w Rozdziale 5.

[12] Przez **słowo znaczące** rozumiem tu słowo, które niesie ze sobą jakąś istotną informację. Słowem znaczącym nie będą np. spójniki, przyimki etc.

[13] Reguły translacji przedstawiam w podrozdziale 4.1.5.

ROZDZIAŁ 4

# Translacja pytania do reprezentacji QQuery

Pierwszym właściwym etapem przetwarzania pytania jest jego **analiza**, która ma doprowadzić do jego **zrozumienia** (patrz definicja 2.4). W rozdziale przedstawiam pojęcie szablonów pytań, które w systemie **Hipisek.pl** służą do translacji pytania do QQuery. Przedstawiam także opis dwóch metod translacji: metodę opartą na szablonach pytań oraz metodę brutalną. Rozdział kończy opis realizacji mechanizmu rozwinięć QQuery oraz sposób oceny gotowych zapytań.

## 4.1. Metoda translacji do QQuery oparta na szablonach pytań

W podrozdziale przedstawiam własny sposób translacji pytań zapisanych w języku naturalnym do pytań reprezentowanych przy pomocy wprowadzonej w Rozdziale 3 struktury QQuery. W tym celu niezbędne jest wykorzystanie podstawowych narzędzi analizy języka naturalnego, których opis rozpoczyna podrozdział. Translacja przebiega w oparciu o **szablony pytań** (ang. *question templates*) i jest realizowana przez moduł o nazwie **QParser** (od ang. *Question Parser*). W podrozdziale zostanie przedstawiona gramatyka szablonów pytań oraz opisane kolejne jej elementy.

### 4.1.1. Rola parsingu i technik NLP w translacji do QQuery

Pierwszym etapem *analizy pytania* jest *analiza syntaktyczna*. Jej celem jest dostarczenie niezbędnych informacji językowych potrzebnych do poprawnej translacji do formalnej reprezentacji QQuery. Proces ten przeprowadzony zostanie za pomocą **parsera** napisanego dla celów niniejszej pracy (dalej nazywanego **parserem Hipisek**). Parser Hipisek przeprowadza:

- **Tokenizację** – podział zdania na poszczególne tokeny,
- **Lematyzację** – sprowadzenie do formy podstawowej i wygenerowanie form fleksyjnych,
- **Synomizację** – dodanie synonimów do zidentyfikowanych wyrazów,
- **Oznaczenie części mowy i dodanie semantyk**,
- **Oznaczenie jednostek nazwanych**.

W następnym paragrafie omawiam w skrócie wykorzystany parser.

### 4.1.2. Opis parsera Hipisek

Parser **Hipisek** jest prostym parserem, powstałym specjalnie dla celów niniejszej pracy. Jego podstawowym elementem jest duży słownik (ok. 120 tys. leksemów) powstały poprzez kompilację ogólnodostępnych źródeł:

- Słownik Języka Polskiego (dawny słownik alternatywny)[1],
- Baza wiedzy **DBPedia**[2],
- Oznaczenie części mowy projektu **Logofag**,[3]
- Słownik synonimów polskich[4],
- Lista imion dopuszczalnych w RP[5],
- Lista miast polskich[6],
- Lista państw świata[7],
- Lista miast w Unii Europejskiej wg liczby ludności[8],
- Lista stolic państw świata[9].

Słownik ten przechowuje informacje o:

- Formach bazowych wyrazu (leksemu),
- Formach odmienionych wyrazu,
- Części mowy wyrazu[10],
- Semantyce wyrazu (np. name, city, place, country ...),
- Frekwencji wyrazu.

Frekwencja wyrazu została przepisana z listy frekwencyjnej, powstałej poprzez zliczenie wyrazów z korpusu stron internetowych, ściągniętych programem typu *crawler* o kodowej nazwie **alef**[11]. Wyrazy rzadkie lub nie występujące na liście frekwencyjnej (np. *abidżanka*) były usuwane ze słownika.

Parser **Hipisek** przeprowadza prostą **lematyzację** przyporządkowując leksem bazowy o najwyższej frekwencji posiadający szukaną formę odmienioną. Nie usuwa on niejednoznaczności i nie rozwiązuje problemu homonimii. Na przykład dla wyrazu *mamy* parser zawsze sprowadza wyraz do formy czasownika *mieć*, choć czasem może to być wyraz oznaczający formę rzeczownika *mama*.

[1] *Słownik języka polskiego* http://www.sjp.pl/

[2] *The DBpedia Knowledge Base* http://dbpedia.org

[3] Projekt **Logofag** polega na zbudowaniu maszyny budującej słowniki wielojęzyczne i realizowany jest przez firmę Poleng Sp. z o.o. http://www.poleng.pl

[4] Słownik synonimów polskich http://elektronikjk.republika.pl/u8.html

[5] Nasz bocian – strony o rodzinie http://www.nasz-bocian.pl

[6] Lista miast opracowanych w AutoMapie XL z pełną informacją nawigacyjną http://www.automapa.com.pl/

[7] Wikipedia: Państwa Świata http://pl.wikipedia.org/wiki/Lista_państw

[8] Wikipedia: Miasta w Unii Europejskiej wg liczby ludności http://pl.wikipedia.org/wiki/Miasta_w_Unii_Europejskiej_według_liczby_ludności

[9] Wikipedia: Stolice państw świata http://pl.wikipedia.org/wiki/Stolice_państw_świata

[10] Tylko część leksemów ma oznaczoną część mowy, która została oznaczona za pomocą prostego skryptu zgadującego część mowy na podstawie bazowej formy wyrazu, bądź przepisana z udostępnionej części projektu **Logofag**.

[11] Patrz Rozdział 2.

Ważnym elementem parsera **Hipisek** jest moduł **NER** (ang. *Named Entity Recognition*), czyli moduł rozpoznający **jednostki nazwane**. Na potrzeby niniejszej pracy przyjmijmy, definicję zawartą w pracy [21][12]:

**Definicja 4.1** ***Jednostką nazwaną*** *nazywamy ciągły fragment tekstu, który odnosi się do jednostek informacji takich jak osoby, miejsca geograficzne, nazwy organizacji lub miejsc, daty, procenty, kwoty pieniężne, miejsca.*

W rozumieniu powyższej definicji **jednostką nazwaną** będzie więc np. fraza *Marcin Walas* oznaczająca w parserze **Hipisek** jednostkę nazwaną typu *person*. Moduł **NER** parsera oznacza takie jednostki nazwane jak:

- Daty,
- Osoby,
- Miejsca.

Język zapisu reguł modułu **NER** został oparty na formalizmie zapisu reguł parsera **Spejd** ([22] i [23]), z pewnymi zmianami zaproponowanymi w projekcie NERT (por. [21]). Jego pełna specyfikacja wraz z przykładowymi regułami znajduje się w Dodatku C niniejszej pracy.

Informacje językowe dostarczone przez parser **Hipisek** stanowią podstawę do translacji pytania do reprezentacji QQuery.

### 4.1.3. Translacja do QQuery w oparciu o szablony pytań

Transfer do QQuery ma za zadanie:

- Zidentyfikowanie typu pytania,
- Zidentyfikowanie tematu pytania,
- Zidentyfikowanie akcji pytania,
- Ekstrakcję ograniczeń,
- Wygenerowanie fraz wyszukujących.

W pracy ograniczę się głównie do sposobu analizy pytania opartego na szablonach pytań (ang. *question templates*). Metody tej używa się np. w pracach: [24], [19] lub [14]. Przyjmijmy, że:

**Definicja 4.2** ***Szablon pytań*** *składa się z* ***wyrażenia dopasowania pytania*** *oraz* ***reguły translacji do QQuery****. Pytanie dopasowuje się do szablonu pytań, jeśli dopasowuje się do* ***wyrażenia dopasowania pytania****.*

[12] Krzysztof Jassem, Filip Graliński, Michał Marcińczuk *„Semi-supervised Learning Rule Acquisition for Named Entity Recognition and Translation"*, tłumaczenie własne.

**Wyrażenie dopasowania pytania** stanowi ciąg warunków jakie muszą spełnić kolejne wyrazy zadanego pytania, by pytanie zostało przyporządkowane do danego szablonu. **Reguły translacji** służą do zbudowania z pytania dopasowanego do wyrażenia dopasowania formalnej reprezentacji QQuery pytania.

Szablon pytań zapisywany jest przy pomocy formalizmu opartego na wspomnianym już formalizmie zapisu reguł parsera **Spejd**([22] i [23]), wraz ze zmianami zaproponowanymi w projekcie NERT (por. [21]). Jego pełna specyfikacja wraz z przykładowymi szablonami znajduje się w Dodatku B niniejszej pracy.

### 4.1.4. Wyrażenie dopasowania pytania

Przyjąłem, że **wyrażenie dopasowania pytania** przeprowadza dopasowanie na poziomie **wyrazu**[13]. Warunek sprawdzany w trakcie dopasowywania wyrażenia dopasowania na danym wyrazie reprezentuję za pomocą **encji dopasowania**. Definicję pojęcia **encji dopasowania** rozpocznę od zdefiniowania podstawowej **encji dopasowania** jaką jest **encja atomowa**:

**Definicja 4.3** ***Encja atomowa*** *– jest to pojedynczy warunek, nakładany na dany wyraz pytania. Encja atomowa* ***jest dopasowana*** *jeśli wyraz spełnia jej warunek.*

W warunkach **atomowych encji dopasowania** można użyć relacji ze zbioru:

$$\{=,\ !=,\ \tilde{},\ !\tilde{}\}$$

Gdzie:

- $=$ – równość,
- $!=$ – różne od,
- $\tilde{}$ – spełnienie wyrażenia regularnego,
- $!\tilde{}$ – niespełnienie wyrażania regularnego.

W ramach przyjętego formalizmu, zdefiniowałem następujące warunki **atomowych encji dopasowania**:

- used $\square$ $x$ – napis jaki wystąpił w pytaniu jest w relacji $\square$ z $x$,
- normal $\square$ $x$ – znormalizowany do małych liter napis, który wystąpił w pytaniu jest w relacji $\square$ z $x$,
- base $\square$ $x$ – forma bazowa wyrazu jest w relacji $\square$ z $x$,
- pos $\square$ $x$ – część mowy wyrazu jest w relacji $\square$ z $x$,
- sem $\square$ $x$ – semantyka wyrazu jest w relacji $\square$ z $x$,
- ne $\square$ $x$ – oznaczenie jednostki nazwanej wyrazu jest w relacji $\square$ z $x$,

[13] Faktycznie jest to dopasowanie na leksemie reprezentującym wyraz, który powstał w etapie analizy pytania przez **parser Hipisek**.

- synonym $\square$ $x$ – forma bazowa jednego z synonimów wyrazu lub forma bazowa wyrazu jest w relacji $\square$ z $x$.

Gdzie $\square \in \{=, != , \tilde{}, !\tilde{}\}$.

☞ **Uwaga** Ze względu na ograniczenia i prostotę **parsera Hipisek** część wyrazów może nie mieć przyporządkowanych pól formy podstawowej, części mowy lub semantyki. W takim przypadku dla relacji pozytywnych (=, ~) zakładamy, że encja **nie jest** spełniona, natomiast dla relacji negatywnych (!=, !~), że encja **jest** spełniona.

Korzystając z definicji **atomowej encji dopasowania** zdefiniuję **encję dopasowania** jako:[14]:

**Definicja 4.4** ***Encją dopasowania*** *nazywamy wyrażenie $E$ spełniające jeden z warunków:*

1. *$E$ jest atomową encją dopasowania*
2. *$E$ jest koniunkcją n (gdzie $n \geqslant 2$) **atomowych encji dopasowania***
3. *$E$ jest alternatywą n (gdzie $n \geqslant 2$) **atomowych encji dopasowania***

W zapisie tekstowym reguły każda encja zostaje otoczona nawiasami trójkątnymi($<\ldots>$). Kolejne warunki w encji oddzielone są średnikiem. Domyślnie warunki łączone są za pomocą koniunkcji. Rozpoczęcie encji od sekwencji znaków **OR:** sprawia, że warunki łączone są za pomocą alternatywy.

Dopasowanie wyrazu do encji następuje zgodnie z definicją 4.5:

**Definicja 4.5** *Jeśli $E$ jest encją dopasowania, natomiast $w$ dowolnym wyrazem, to mówimy że $w$ **dopasowuje się** do $E$ jeśli spełniony jest jeden z warunków:*

1. *Jeśli $E$ jest encją atomową, to $w$ spełnia warunek zawarty w $E$.*
2. *Jeśli $E$ jest alternatywą encji $E_1, E_2 \ldots E_n$, to $\exists_{1 \leqslant i \leqslant n} w$ dopasowuje się do $E_i$.*
3. *Jeśli $E$ jest koniunkcją encji $E_1, E_2 \ldots E_n$, to $\forall_{1 \leqslant i \leqslant n} w$ dopasowuje się do $E_i$.*

☞ **Uwaga** Encje można potraktować jako rozszerzone **wyrażenia regularne**, którym dodano możliwość dopasowania w oparciu o informacje lingwistyczne (część mowy, forma bazowa) i semantyczne.

Przykładowe **encje dopasowania szablonu pytań** wraz z komentarzem znajdują się w tabeli 4.1.

Ponadto każdej z encji możemy dodać jeden z modyfikatorów:

- {n,m} – wystąpienie co najmniej n razy oraz co najwyżej m razy (podobnie jak operator {n,m} w wyrażeniach regularnych języka **Perl**)

[14]Można wprowadzić bogatszy język encji dopasowania zawierający np. negację, bogatsze koniunkcje i alternatywy (np. do klasycznego rachunku zdań), ale należy mieć na uwadze możliwości zastosowania tworzonego modelu i ograniczenia związane ze złożonością obliczeń.

Tabela 4.1: Przykładowe encje dopasowania szablonu pytań

| Lp. | Encja dopasowania | Komentarz |
|---|---|---|
| 1. | <used˜[a-z]+> | Spełniona przez każdy wyraz pisany małymi literami alfabetu łacińskiego |
| 2. | <pos!=subst> | Spełniona przez każdy wyraz o części mowy różnej od rzeczownika lub niezdefiniowanej części mowy |
| 3. | <base˜[ab].*> | Spełniona dla wyrazów o formie bazowej zaczynającej się od małej litery **a** lub **b** |
| 4. | <sem=name; used˜.*a> | Spełniona dla wyrazów o semantyce **name** oraz kończących się na literę a |
| 5. | <OR:pos=subst;pos=verb> | Spełniona dla wyrazów będących rzeczownikami **lub** czasownikami |
| 6. | <pos˜(?:subst\|verb)> | Inny zapis encji nr 5 |

- * – wystąpienie dowolną ilość razy (także zero)
- + – wystąpienie dowolną (niezerową) ilość razy
- ? – wystąpienie zero lub jeden raz
- TRUE – zawsze spełniona

Ze względu na spełnienie warunku modyfikatora mówimy, że:

**Definicja 4.6** ***Encja dopasowania*** *z modyfikatorem* $modif$ *jest* ***spełniona*** *jeśli dla danego ciągu wyrazów* $(w)$*:*

1. *Zawsze, dla* $modif = TRUE$*,*
2. *Jest dopasowana do* $n$ *pierwszych wyrazów ciągu* $(w)$ *gdzie:*
   - *(a)* $n = 1$*, dla niezdefiniowanego modyfikatora* $modif$*,*
   - *(b)* $n \in [0, 1]$*, dla* $modif \equiv ?$*,*
   - *(c)* $n \in [0, \infty]$*, dla* $modif \equiv *$*,*
   - *(d)* $n \in [1, \infty]$*, dla* $modif \equiv +$*,*
   - *(e)* $n \in [s, t]$*, dla* $modif \equiv \{s, t\}$*.*

Mając definicję **encji dopasowania** mogę zdefiniować **wyrażenie dopasowania pytania jako**:

**Definicja 4.7** ***Wyrażenie dopasowania pytania*** *to ciąg* $n$ *(gdzie* $n > 0$*)* ***encji dopasowania****.*

Dopasowanie **wyrażenia dopasowania pytania** następuje zgodnie z definicją:

**Definicja 4.8** ***Wyrażenie dopasowania pytania*** *będące ciągiem* ***encji dopasowania*** $(e_0, e_1, e_2 \dots e_n)$ *jest* ***dopasowane*** *jeśli dla pytania, rozumianego jako ciąg wyrazów* $(w_0, w_1, w_2 \dots w_m)$ *istnieje takie dopasowanie encji* $e_i$*, że* $\forall_{i \in [0,n]} e_i$ *jest spełniona.*

Zgodnie z zaprezentowaną definicją szablonu pytania (4.2), jeśli **wyrażenie dopasowania pytania** jest dopasowane to **szablon pytania** jest dopasowany. W przypadku dopasowania **szablonu pytania** możemy przetłumaczyć dopasowane pytanie do formalnej reprezentacji QQuery za pomocą **reguły translacji**. Przejdę teraz do opisu **reguł translacji**.

### 4.1.5. Reguły translacji do QQuery

Przyjąłem, że każdemu ze zdefiniowanych **szablonów pytań** należy przyporządkować **regułę translacji**. Przyjmijmy, że:

**Definicja 4.9** *Reguła translacji to instrukcja wypełnienia danymi własności QQuery (typ, temat, akcja, ograniczenia, frazy) na podstawie encji dopasowanych za pomocą wyrażenia dopasowania pytania.*

Reguła translacji składa się z następujących elementów:

- **Type** – definicja typu pytania pasującego do szablonu,
- **Topic** – instrukcja, co jest **tematem** pytania,
- **Action** – instrukcja, co jest **akcją** pytania,
- **Constraints** – instrukcja, jakie **ograniczenia** należy dołożyć do pytania, lub które z leksemów wchodzących w skład pytania nie mogą trafić do zbioru ograniczeń,
- **Phrase** (może pojawić się wielokrotnie) – instrukcja wygenerowania **fraz wyszukujących**.

W celu jaśniejszego omówienia kolejnych elementów reguły translacji rozpatrzmy przykładowy szablon pytania postaci[15]:

```
# ===============
# Przykładowy szablon pytań

Question: sample

# Wyrażenie dopasowania
Match: <normal=podaj>? <normal=gdzie> <normal=jest> <ne~person>+

# Reguła translacji:
Type:        PLACE;PRESENT
Topic:       \4
```

[15] Szablon ten został zbudowany dla celów poglądowych i nie występuje w docelowej implementacji systemu **Hipisek.pl**

```
Action:      lexeme(być)
Constraints: -\1;lexeme(teraz)
Phrase:      used(\topic); przebywa|jest; w
Phrase:      used(\topic); poleciał|pojechał; do|na
```

Szablon taki dopasuje takie pytania jak:

1. Gdzie jest Lech Kaczyński?
2. Podaj gdzie jest Donald Tusk.
3. Gdzie jest Jan Maria Rokita?

Analizując ten przykład, omówię teraz kolejne elementy reguły translacji.

### Sekcja type

Ta część reguły translacji składa się z dwóch elementów:

- Typu odpowiedzi (TIME, PLACE, UNKNOWN),
- Charakteru wymaganej wiedzy (PAST, PRESENT, FUTURE, UNKNOWN).

Typy podajemy w regule translacji literalnie w postaci pary dopuszczalnych wartości. W rozpatrywanym przykładzie pytaniu zostanie przyporządkowany typ (PLACE, PRESENT), tzn. jest to pytanie o miejsce (PLACE), którego odpowiedź ma odzwierciedlać stan aktualny (PRESENT).

### Sekcja topic

W tej sekcji można umieścić definicję co jest tematem dopasowanego pytania. Możliwe jest wpisanie do tematu dowolnej sumy[16] spośród poniższych wartości:

- Leksem ze słownika o formie bazowej $x$,
- Leksem (leksemy) dopasowane do elementu nr $n$ encji dopasowania.

W rozpatrywanym przykładzie do tematu zostaje wpisane leksemy dopasowane do czwartej encji dopasowania szablonu. Dla rozpatrywanych pytań będą to więc leksemy:

1. Lech, Kaczyński
2. Donald, Tusk
3. Jan, Maria, Rokita

### Sekcja action

Pole **action** jest analogicznie budowane do pola **topic** opisanego wyżej. W rozpatrywanym przykładzie wpisujemy zawsze do akcji leksem ze słownika o formie bazowej „być”[17]. Każde z dopasowanych pytań będzie więc miało tą samą akcję, będącą pojedynczym leksemem *być*.

[16] Temat może być ciągiem leksemów dlatego w przyjętym formalizmie umożliwiam podanie kilku opcji.

[17] Zauważmy, że identyczny efekt uzyskano by, gdyby posłużono się konstrukcją \3, podobną do tej użytej w sekcji **topic**. Konstrukcja *lexeme* została tu użyta dla celów poglądowych.

**Sekcja constraints**

Zbiór ograniczeń QQuery budowany jest na dwa sposoby:

1. Poprzez dodanie znaczących słów niewykorzystanych w polach **topic** lub **action**.
2. Poprzez dodanie ograniczeń zdefiniowanych w szablonie pytań.

Pierwszy z tych sposobów realizowany jest za pomocą prostego algorytmu:

**Algorytm 4.1** ***Algorytm tworzenia zbioru ograniczeń QQuery***

1. *Niech W oznacza zbiór wszystkich leksemów przetwarzanego pytania.*
2. *Ze zbioru W usuń wszystkie leksemy przypisane do pól **action** i **topic** tworzonego QQuery.*
3. *Ze zbioru W usuń leksemy, które wystąpiły w postaci wyrazu o długości nie przekraczającej progu $c_{min}$.*
4. *Ze zbioru W usuń wszystkie leksemy, które w słowniku oznaczone zostały komentarzem `NOCONSTRAINT`[18].*
5. *Ze zbioru W usuń wszystkie leksemy, których forma bazowa lub postać znormalizowana (normal) znajduje się w specjalnym zbiorze zablokowanych grafemów (zbiór ten tworzony jest ręcznie i zawiera najczęstsze wyrazy, które występują w pytaniu a nie mają charakteru ograniczeń, np.: podaj, gdzie, kiedy etc.).*
6. *Tak powstały zbiór W przypisz jako zbiór ograniczeń tworzonego QQuery.*

Do realizacji drugiego sposobu służy sekcja **constraints** reguły translacji. Za jej pomocą można:

- Zdefiniować nowe ograniczenie w postaci słowa nie występującego w pytaniu,
- Zablokować jedno ze słów dopasowanych w encji dopasowania, tak aby nie trafiło do zbioru ograniczeń.

W rozpatrywanym przykładzie ze zbioru ograniczeń usuwamy pierwszy dopasowany token (wyraz *„podaj”*), natomiast dodajemy dodatkowe ograniczenie w postaci wyrazu o formie bazowej *„teraz”*. Wyraz *„gdzie”* nie trafi do zbioru ograniczeń, gdyż został mu przypisany komentarz `NOCONSTRAINT`[19].

**Sekcja phrase**

Ostatnią z sekcji reguły translacji do QQuery jest sekcja **phrase**, zawierająca informację o wygenerowaniu fraz wyszukujących odpowiedzi na zadane pytanie. Frazy takie są ciągami tekstu, które z dużym prawdopodobieństwem występują w artykule źródłowym *blisko* tekstu zawierającego odpowiedź na zadane pytanie. Dla pytania:

Gdzie jest Donald Tusk?

Frazami wyszukującymi mogą być na przykład:

[18] Stworzony przeze mnie słownik pozwala dodawać komentarze do leksemów, służy do tego pole **remark**.

[19] Faktycznie taki sam komentarz jest też przypisany do wyrazu *„podaj”*, dlatego dodane w polu **constraints** ograniczenie było w tym przypadku zbędne.

- Donald Tusk przebywa w
- Donald Tusk poleciał do
- Donald Tusk jest w

Sekcja **phrase** składa się z ciągu **tokenów** dodawanych do frazy po kolei. Każdy token może być:

- tekstem
- zbiorem tekstów (oddzielonych znakiem |)
- jednym z pól: used, base, normal, form, synonyme dowolnego z leksemów spośród jednego z poniższych:
  - Leksem ze słownika o formie bazowej $x$
  - Leksem (leksemy) dopasowane do elementu nr $n$ encji dopasowania
  - Leksem (leksemy) oznaczone jako temat lub akcja QQuery

W rozpatrywanym przykładzie dla pytania: *Gdzie jest Lech Kaczyński?* przez pierwszą sekcję **phrase** wygenerowane zostaną frazy postaci:

- Lech Kaczyński przebywa w
- Lech Kaczyński jest w

Druga z nich wygeneruje natomiast frazy:

- Lech Kaczyński poleciał do
- Lech Kaczyński poleciał na
- Lech Kaczyński pojechał do
- Lech Kaczyński pojechał na

☞ **Uwaga** Nie wszystkie pola QQuery muszą zostać wypełnione. Część z nich może zostać pusta, co niekoniecznie musi w negatywny sposób wpłynąć na dalsze przetwarzanie pytania. Dotyczy to w szczególności pól **type** oraz **constraints**.

Formalna definicja języka zapisu szablonów pytań wraz z przykładowymi szablonami znajduje się w dodatku B niniejszej pracy.

## 4.2. Brutalna metoda translacji do QQuery

W przypadku, gdy żaden ze zdefiniowanych szablonów pytań nie dopasuje się do przetwarzanego pytania, następuje **brutalne** dopasowanie pytania do QQuery. W tym celu użyłem następującego prostego algorytmu:

**Algorytm 4.2** ***Algorytm brutalnej translacji do QQuery***

1. *Każdemu leksemowi nadaj wagę $w_a$ bycia akcją pytania*

2. *Wybierz leksem o najwyższej wadze $w_a$ i przypisz go do pola **action** QQuery*
3. *Każdemu leksemowi (poza wybranym w punkcie 2.) nadaj wagę $w_t$ bycia tematem pytania*
4. *Wybierz leksem o najwyższej wadze $w_t$ i przypisz go do pola **topic** pytania*
5. *Spróbuj przewidzieć typ odpowiedzi na pytanie $t_t$ na podstawie jego frazy rozpoczynającej*
6. *Dodaj do zbioru **ograniczeń** QQuery wszystkie niewykorzystane leksemy wg algorytmu dodawania ograniczeń (patrz algorytm 4.1)*

Waga $w_a$ leksemu obliczana jest jako suma ważona elementów wektora, którego kolejne elementy przyjmują następujące wartości:

- Czy zdefiniowano pos leksemu i czy pos = verb? (1 jeśli warunek jest spełniony, 0 w p.p.),
- Czy semantyka leksemu jest pusta? (1 jeśli warunek jest spełniony, 0 w p.p.),
- Ważona odległość od początku pytania (im dalej od początku, tym waga mniejsza).

Natomiast waga $w_t$ liczona jest jako suma ważona elementów wektora, którego kolejne elementy przyjmują następujące wartości:

- Czy zdefiniowano pos i czy pos = subst? (1 jeśli warunek jest spełniony, 0 w p.p.),
- Czy pos != adj i pos != verb? (1 jeśli warunek jest spełniony, 0 w p.p.),
- Czy zdefiniowano sem? (1 jeśli warunek jest spełniony, 0 w p.p.),
- Czy leksem oznaczono jako jednostkę nazwaną? (1 jeśli warunek jest spełniony, 0 w p.p.),
- Czy wyraz pisany jest wielką literą? (1 jeśli warunek jest spełniony, 0 w p.p.),
- Ważona odległość od początku pytania (im dalej od początku, tym waga mniejsza).

## 4.3. Rozwinięcie QQuery

Rozwijanie QQuery to proces polegający na dodaniu informacji do zapytania, które mogą pomóc w znalezieniu poprawnej odpowiedzi na zadane pytanie. W dziedzinie QA stosuje się wiele, często bardzo wysublimowanych technik dodania informacji, takich jak:

- Zastosowanie ontologii do identyfikacji relacji między pojęciami (por. [25])
- Dodanie informacji na poziomie języka (np. generowanie form fleksyjnych por. [20])

W mojej pracy ograniczyłem się do rozwinięcia QQuery w oparciu o informacje na poziomie języka. Rozwinięcie takie polega na:

- dodaniu form fleksyjnych – realizowanego przez parser **Hipisek**,
- dodaniu synonimów – realizowanego przez parser **Hipisek**,[20]
- dodaniu fraz wyszukujących – realizowanego na etapie translacji za pomocą szablonów pytań.

[20]Synonimy są w systemie **Hipisek.pl** czym innym niż w tradycyjnym ujęciu ontologicznym. Synonim to pewnego rodzaju *forma wariantywna* (w sensie napisu), czyli inny napis, który może oznaczać to samo, lub *prawie* to samo co analizowany wyraz.

## 4.4. Ocena trafności QQuery

Ocena trafności QQuery ma na celu wskazanie, które informacje są istotne z punktu widzenia poszukiwania odpowiedzi na zadane pytanie, oraz wskazaniu tych, które potencjalnie generować mogą błędne wyniki. Dlatego każdemu wyrazowi lub frazie występującym w ramach pól: temat, akcja, ograniczenia jest przyporządkowana liczba rzeczywista $p \in [0, 1]$. Im liczba $p$ jest większa, tym jakość danego wyrazu jest wyższa. Liczbę $p$ nazwiemy **oceną**.

### 4.4.1. Oceny pól tematu i akcji

Pola tematu i akcji przypisane za pomocą metody dopasowania szablonu zawsze otrzymują ocenę najwyższą równą 1. Przyjmujemy bowiem, że translacja tą metodą jest dokładna.

W przypadku użycia algorytmu brutalnej translacji pola te otrzymują ocenę równą przeskalowaniu wagi $w_x$ (gdzie $x$ oznacza odpowiednio $t$ dla tematu QQuery oraz $a$ dla akcji QQuery) na zbiór $[0, 1]$. Przeskalowanie to zostaje przeprowadzone w następujący sposób:

Niech $min(w_x)$ oznacza minimalną wagę $w_x$ dla leksemów z przetwarzanego pytania, $canmax(w_x)$ maksymalną możliwą do osiągnięcia wagę $w_x$ dla leksemów z przetwarzanego pytania, $factor \in [0, 1]$ oznacza współczynnik skalujący metodę brutalną (będący maksymalną możliwą dla niej oceną) wtedy ocenę liczymy poprzez zastosowanie wzoru:

$$h(w_x) = factor * \frac{w_x - min(w_x)}{canmax(w_x) - min(w_x)}$$

W przypadku gdy $canmax(w_x) = min(w_x)$ ocena $h(w_x)$ jest równa $1$.

Współczynnik $factor$ dobrany został heurystycznie i ustawiony na wartość $0.9$.

### 4.4.2. Oceny ograniczeń

W przypadku oceniania ograniczeń bierzemy pod uwagę ich frekwencję. Przyjmujemy, że ograniczenia o niższej frekwencji są bardziej istotne z punktu widzenia dalszego przetwarzania, niż ograniczenia mające wysoką frekwencję. Oceny ograniczeń dokonujemy w następujący sposób:

Niech $f_{min}$ oznacza minimalną frekwencję leksemu spośród wchodzących w skład zbioru ograniczeń przetwarzanego pytania, $f_{max}$ maksymalną frekwencję leksemu spośród wchodzących w skład zbioru ograniczeń przetwarzanego pytania, natomiast $f(x)$ frekwencję leksemu $x$. Wówczas ocenę ograniczenia leksemu $x$ liczymy ze wzoru:

$$h(x) = \frac{f_{max} - f(x)}{f_{max} - f_{min}}$$

W przypadku, gdy $f_{max} = f_{min}$ ocena $h(x)$ jest równa $1$.

ROZDZIAŁ 5

# Metody wyszukiwania odpowiedzi w tekście niesformatowanym

W rozdziale przedstawiam metody wyszukiwania potencjalnych odpowiedzi w bazie artykułów prasowych zapisanych w postaci *litego tekstu*. Artykuły zorganizowane zostają w specjalnej **bazie wiedzy**. Rozdział rozpoczyna opis bazy wiedzy i procesu indeksowania artykułów. Następnie omawiam trzy metody wyszukiwania odpowiedzi stworzone na potrzeby systemu **Hipisek.pl** oraz sposób ich zaimplementowania w ramach **wyszukiwarki odpowiedzi**. Rozdział kończy opis sposobu oceny uzyskanych artykułów.

## 5.1. Baza wiedzy

Baza wiedzy stanowi repozytorium artykułów prasowych, będących źródłem odpowiedzi dla budowanego systemu QA **Hipisek.pl**. Dla uproszczenia możemy traktować bazę wiedzy jako zbiór plików tekstowych opatrzonych pewnymi atrybutami w postaci **metainformacji**.[1] Przyjmijmy:

**Definicja 5.1** ***Artykuł*** *to pojedynczy tekst pobrany z danego źródła zapisany w postaci tekstu niesformatowanego.*

Artykuły indeksuję za pomocą robota internetowego typu *crawler*. Zaindeksowane artykuły, które raz trafiły do bazy wiedzy, pozostają tam w niezmienionej formie (nawet jeśli artykuł źródłowy został zaktualizowany).

Należy dodać, że użycie w bazie wiedzy **metainformacji** choć istotne, nie jest głównym problemem poruszanym w ramach niniejszej pracy. Źródłem odpowiedzi budowanego systemu QA jest *tekst niesformatowany*. Użycie **metainformacji** jest naturalnym rozwinięciem metod czysto statystycznych (omawianych w ramach mojej pracy) w kierunku metod bazujących na bardziej sformalizowanym ujęciu wiedzy i jej postaci (nie poruszanych w ramach niniejszej pracy). Porównaj np. baza wiedzy w postaci faktów języka Prolog w pracy [26].

### 5.1.1. Metainformacje

Choć budowany system QA, jako źródło danych przyjmuje tekst w postaci niesformatowanej, to *de facto* działa na jego pewnej reprezentacji formalnej, będącej wynikiem wyodrębnienia istotnych **metainformacji** zawartych w tekście i zapisania ich w bazie wiedzy. Proces ten przebiegać będzie w momencie **indeksowania**[2] i przeprowadzony będzie dla danego artykułu tylko raz.

Dla danego **artykułu** wyróżniłem następujące **metainformacje**[3]:

[1] Pojęcie **metainformacji** przedstawiłem w Rozdziale 2.

[2] O indeksowaniu traktuje paragraf 5.1.2 niniejszego rozdziału.

[3] Uwaga w nazwach **metainformacji** zachowałem oryginalną pisownię z implementacji konkretnych rozwiązań, stąd pojawiające się nazwy angielskie.

- **url** – jest to adres internetowy, z którego pobrano dany artykuł,
- **title** – jest to tytuł, pod jakim udostępniony był w danym serwisie prasowym dany artykuł,
- **source** – źródło artykułu, czyli serwis prasowy, z którego został pobrany artykuł,
- **stamp** – data opublikowania artykułu,
- **md5** – suma md5 znormalizowanego tekstu artykułu, wygenerowana za pomocą programu md5sum[4].

Proces automatycznego pozyskiwania **metainformacji** przedstawiam w paragrafie 5.1.4 niniejszego podrozdziału.

### 5.1.2. Indeksowanie artykułów

**Definicja 5.2** ***Indeksowaniem artykułu*** *nazywamy automatyczny proces polegający na wczytaniu treści artykułu prasowego, wyekstrahowaniu z niej niezbędnych metainformacji i zapisaniu artykułu w bazie wiedzy.*

Indeksowaniem artykułów zajmuje się specjalnie zbudowany program indeksujący. Program ten korzysta z dodatkowej bazy danych (tzw. *bazy linków* o nazwie **Aleksandria**[5]), w której przechowuje wszystkie linki znalezione w ramach przetwarzanych witryn serwisów prasowych, będące potencjalnymi kandydatami na *„bycie artykułem"*[6]. Baza **Aleksandria** przechowuje następujące informacje dotyczące zebranych linków:

- **url** – znaleziony link (adres url),
- **state** – stan linku (nieprzetworzony lub przetworzony),
- **indexed** – stan zaindeksowania linku (niezaindeksowany lub zaindeksowany),
- **md5** – suma md5 znormalizowanej zawartości przetworzonego linku (dla wyeliminowania duplikatów).

Mówimy, że link może być w następujących stanach:

- **Przetworzony** – jeśli została pobrana strona internetowa, do której prowadzi,
- **Nieprzetworzony** – jeśli strona nie została pobrana, czyli link został tylko znaleziony i zapisany do późniejszego przetworzenia,
- **Zaindeksowany** – jeżeli rozpoznano, że link jest artykułem i zapisano go do bazy wiedzy,

[4]**Suma md5** jest tzw. sumą kontrolną pliku. Program md5sum dla każdego pliku generuje specjalną *unikatową* liczbę, którą później można traktować jak jednoznaczny identyfikator pliku.

[5]Na wzór słynnej antycznej biblioteki Aleksandryjskiej.

[6]Przypominam, że proces indeksowania jest w pełni automatyczny, stąd konieczność automatycznego pozyskania linków potencjalnych artykułów.

- **Niezaindeksowany** – jeżeli **przetworzony** link nie został jeszcze zapisany do bazy wiedzy.

W procesie indeksowania artykułu można wyróżnić następujące etapy:

1. **Roboting** – pobranie treści dokumentu ze strony www i pozyskanie linków do kolejnych artykułów,
2. **Extracting** – znormalizowanie treści dokumentów, wyeliminowanie błędów technicznych oraz ekstrakcja metainfomacji z artykułu,
3. **Writing** – zapis artykułu do bazy wiedzy.

Proces indeksowania dokumentu następuje automatycznie i jest uruchamiany co pewien określony czas (ma to na celu wczytanie do bazy wiedzy pojawiających się, nowych artykułów). W następnych paragrafach omówię kolejne etapy indeksowania artykułu.

Wynikiem algorytmu jest zbiór zapisanych w bazie wiedzy nowych artykułów, gotowych do przetwarzania przez wyszukiwarkę odpowiedzi. Ubocznym skutkiem jego działania jest zwiększenie ilości znalezionych adresów url w ramach bazy **Aleksandria**.

### 5.1.3. Robot indeksujący

Robot indeksujący (o nazwie **Aleksandria**) jest programem, którego głównym zadaniem jest pobranie stron prowadzących do artykułów prasowych umieszczonych w ramach przetwarzanych serwisów prasowych. Realizuje on więc pierwszy etap indeksowania artykułu (etap *roboting*). Pobraną stronę nazwiemy **dokumentem** lub **dokumentem HTML**.

**Definicja 5.3** ***Dokument*** *– (dokument HTML) to plik tekstowy stanowiący źródło strony o danym adresie url, będącym jedną z podstron przetwarzanych serwisów prasowych.*

Pierwszym problemem, z jakim należy się zmierzyć, tworząc program pozyskujący artykuły do zaindeksowania, jest ograniczenie przeszukiwanych stron internetowych tylko do stron przetwarzanych serwisów prasowych. W tym celu przyjmiemy, że każdy adres url serwisu prasowego definiuje **witrynę**. Przyjmijmy, że:

**Definicja 5.4** *Do* ***witryny*** *o adresie url $w$ należy każda strona o adresie url $u$ spełniającym przynajmniej jeden z warunków:*

1. *Jeśli traktujemy adresy $w$ i $u$ jako struktury katalogowe, to $u$ jest podkatalogiem dla $w$* ***lub*** *$u$ jest równe $w$.*
2. *Jeśli z domeny adresu $w$ lub $u$ usuniemy lub dodamy przedrostek www, to spełniony jest warunek (1).*
3. *Jeśli przedrostek www z domeny $w$ lub $u$ zastąpimy pierwszym członem struktury katalogowej znajdującej się po domenie adresu, to spełniony jest warunek (1).*
4. *Jeśli pierwszy człon adresu domenowego zastąpimy przedrostkiem www lub przedrostkiem pustym oraz wstawimy przed pierwszy element struktury katalogowej adresu $u$ lub $w$* ***po*** *domenie, to spełniony jest warunek (1).*

W rozumieniu powyższej definicji do witryny należą wszystkie podstrony strony o adresie $w$ (warunek 1), strony o domenie skróconej lub rozszerzonej (czyli odpowiednio bez lub z prefiksem **www**, warunek 2) lub strony o odpowiedniej subdomenie (warunki 3 i 4). Przykłady adresów zaklasyfikowanych do danej witryny zawiera tabela 5.1.

Tabela 5.1: Przykłady przyporządkowania wybranych adresów url do witryn, dla zbioru witryn składającego się z adresów: www.hipisek.pl, katalog.ploteczka.com oraz hipisek.com.

| Adres url | Przyporządkowana witryna | Spełniony warunek |
|---|---|---|
| www.hipisek.pl/strona.html | www.hipisek.pl | 1 |
| hipisek.pl/strona.html | www.hipisek.pl | 2 |
| www.ploteczka.com/katalog | katalog.ploteczka.pl | 3 |
| katalog.hipisek.pl | www.hipisek.pl | 4 |
| www.ploteczka.com | nie zaakceptowana | brak |
| katalog.hipisek.com | hipisek.com | 4 |

Robot **Aleksandria** działa tylko na stronach pochodzących ze zdefiniowanego zbioru witryn (w tym przypadku witryn przetwarzanych serwisów prasowych). Dzięki temu program pobiera strony tylko znajdujące się w ramach wybranego serwisu i nigdy nie przetwarza stron spoza tego zbioru.

Uproszczony algorytm działania robota **Aleksandria** znajduje się poniżej. Robot ten jest rozbudowaną i znacznie rozszerzoną wersją robota internetowego **alef** napisanego we współpracy z doktorem Filipem Gralińskim[7]. Pewne szczegóły implementacyjne i rozwiązania techniczne zaczerpnięto z książki [27].

**Algorytm 5.1** ***Uproszczony algorytm pobierania stron z serwisów prasowych za pomocą robota Aleksandria***

1. *Niech dany będzie zbiór adresów internetowych źródeł artykułów prasowych $S$, zbiór witryn $W$ serwisów ze zbioru $S$ oraz pusty zbiór $URLS$.*
2. *Do zbioru $URLS$ dodaj wszystkie linki ze zbioru $S$.*
3. *Ustaw flagę* `take_all` *na wartość* `TRUE`*.*
4. *Dla każdego linku $u$ ze zbioru $URLS$ wykonaj następujące kroki:*
   (a) *Pobierz zawartość strony internetowej, do której prowadzi link $u$.*
   (b) *Wyekstrahuj z pobranej strony HTML zawartość tekstową.*
   (c) *Znajdź wszystkie linki, znajdujące się na pobranej stronie, które prowadzą do stron w ramach witryn zbioru $W$ i zapisz je do zbioru $L$.*
   (d) *Jeśli flaga* `take_all` ***nie jest*** *ustawiona to usuń ze zbioru $L$ wszystkie linki, których adres url nie spełnia wyrażenia regularnego typowego adresu url artykułu danego serwisu.*
   (e) *Usuń ze zbioru $L$ wszystkie linki, które już znajdują się w bazie* ***Aleksandria****.*
   (f) *Sprawdź czy $u$ jest artykułem, jeśli tak to sprawdź za pomocą jego sumy md5 oraz sum md5 artykułów zapisanych w bazie, czy $u$ nie został już zaindeksowany.*

[7] O programie tym wspomniałem w Rozdziale 2. Posłużył do pozyskania korpusu tekstów internetowych.

(g) *Jeśli $u$ nie został zaindeksowany, to zapisz tekst artykułu $u$ do bazy **Aleksandria**.*

(h) *Jeśli $u$ jest artykułem **lub** flaga `take_all` jest ustawiona to zapisz wszystkie linki ze zbioru $L$ do bazy **Aleksandria** jako nieprzetworzone i niezaindeksowane.*

(i) *Zapisz link $u$ do bazy **Aleksandria** jako przetworzony.*

5. *Pobierz z bazy **Aleksandria** kolejną porcję nieprzetworzonych linków do przetworzenia.*

6. *Ustaw flagę `take_all` na wartość **FALSE***

7. *Jeśli nie upłynął limit czasu działania programu to przejdź do kroku 4 algorytmu*

W kolejnych krokach algorytmu robot pobiera tekst interesujących nas stron (zapisując tylko strony będące artykułami) oraz pobiera wszystkie linki potencjalnie kierujące do stron internetowych. W pierwszym kroku robot jest **zachłanny** tzn. zapisuje wszystkie linki w ramach przetwarzanych witryn. Związane jest to z problemem linków opartych na mechanizmie **przekierowania**. Dla przykładu w serwisie prasowym **www.tvn24.pl** na stronie głównej pojawia się link do artykułu postaci:

www.tvn24.pl/3a5-49b96546

Link ten w rzeczywistości (korzystając z mechanizmu przekierowania) prowadzi do artykułu o adresie:

www.tvn24.pl/-1,1590500,0,1,malysz-najdalej,wiadomosc.html

Aby poradzić sobie z wystąpieniem tego typu sytuacji zastosowałem pewne ustępstwo poprzez wprowadzenie flagi **take_all**. Flaga ta powoduje pobranie z sieci wszystkich stron znalezionych w serwisie prasowym, a następnie ich przetworzenie (jednak nadal tylko artykuły zostaną zapisane w bazie). Po pierwszym przebiegu algorytmu flaga jest ustawiana na wartość `false`, przez co mechanizm ten zostaje wyłączony, aż do ponownego uruchomienia skryptu.[8].

Ważną własnością algorytmu jest pobieranie linków tylko z witryn przetwarzanych serwisów prasowych, oraz (w kolejnych krokach) tylko linków prowadzących do artykułów prasowych. Zakładam, że wszystkie adresy url artykułów danego serwisu spełniają pewne wyrażenie (wyrażenia) regularne, np. dla serwisu **www.tvn24.pl** są to adresy zakończone frazą:

```
,wiadomosc.html
```

Znalezione przez robota strony są unikatowe i po zakończeniu działania programu trafiają do programu czyszczącego i przygotowującego je do właściwego zaindeksowania.

### 5.1.4. Czyszczenie pobranych dokumentów i ekstrakcja metainformacji

Czyszczenie jest pierwszą fazą etapu **extracting** procesu indeksowania artykułów. Czyszczenie pobranych dokumentów polega na:

- Usunięciu błędnie pobranych artykułów,

[8]Oczywiście opisany tu problem jest specyficzny dla przetwarzanego serwisu. Prawdopodobnie rozszerzenie zbioru serwisów prasowych spowodowałoby wystąpienie kolejnych problemów technicznych, specyficznych dla technologii tworzenia danego serwisu. Z punktu widzenia niniejszej pracy jest to jednak problem czysto techniczny, którego nie będę dokładniej analizował.

- Wyodrębnieniu treści artykułu z pobranego dokumentu (poza treścią, robot pobiera także takie tekstowe elementy jak, np. komentarze użytkowników serwisu),
- Wyodrębnieniu z artykułu **metainformacji** zapisanych w artykule w sposób jawny (data opublikowania, tytuł i zdefiniowane przez serwis prasowy słowa kluczowe),
- Usunięciu z treści artykułu informacji nieistotnych (np. podpisów pod zdjęciem),
- Normalizacji treści artykułu do formatu: w każdej linii osobne zdanie.

Pierwsze z wymienionych zadań jest pewnego rodzaju sprawdzeniem zebranego materiału. Skrypt czyszczący, dokonuje na tym etapie sprawdzenia czy:

- Strona artykułu została pobrana poprawnie,
- Strona artykułu nie zawiera błędów.

W przypadku wykrycia jednej z powyższych nieprawidłowości, pobrany artykuł jest usuwany.

Wyodrębnienie treści artykułu polega na analizie budowy dokumentu HTML reprezentującego artykuł. Przyjąłem założenie, że:

**Założenie 5.1** *Każdy artykuł prasowy znajdujący się w ramach danego serwisu prasowego ma taką samą strukturę HTML.*

Oznacza to, że każdy z tych artykułów jest generowany w oparciu o pewien specyficzny dla danego serwisu szablon. Ekstrakcję treści artykułu można więc dzięki temu zrealizować jako odnalezienie konkretnego fragmentu w ramach odpowiedniego szablonu. Zadanie to rozwiązałem za pomocą serii **wyrażeń regularnych**.

Dla każdego z przetwarzanych serwisów prasowych określiłem szablon (a raczej jego uproszczenie niezbędne do moich celów), według którego generowane są artykuły. Szablon ten składa się z szeregu wyrażeń regularnych, które identyfikują, czy dana partia tekstu jest jedną z interesujących mnie informacji (treścią artykułu, datą opublikowania, słowami kluczowymi). Każdy pobrany dokument został poddawany skryptowi, który dopasowywał go do szablonu odpowiadającego serwisowi prasowemu. W ten sposób z pobranej treści całego dokumentu uzyskałem dokładnie określone: treść artykułu, datę opublikowania, słowa kluczowe (jeśli takowe wystąpiły w artykule).

Dla przykładu **słowa kluczowe** w artykułach serwisu **www.pudelek.pl** znajdują się zawsze w osobnej linii rozpoczynającej się sekwencją: **Słowa kluczowe:**. Skrypt wykrywał takie linie, odcinał zbędny tekst jej początku, a pozostałość traktował jako listę gotowych słów kluczowych (oddzielonych przecinkami). Fragment treści dokumentu HTML (czyli dokumentu HTML pozbawionego znaczników) ilustrujący podany przykład znajduje się w tabeli 5.2.

W podobny sposób dokonałem ekstrakcji pozostałych **metainformacji**, czyli tytułu oraz daty opublikowania.

☞ **Uwaga** Oczywiście opisywana metoda ma kilka bardzo poważnych technicznych wad. Podstawowym problemem byłaby zmiana szablonu, w ramach którego umieszczane są artykuły. Również ewentualne dodanie obsługi kolejnych serwisów prasowych byłoby trudne (wymagałoby opracowania kolejnych szablonów). Nie jest to jednak problem istotny ze względu na temat niniejszej pracy.

Tabela 5.2: Fragment pobranej treści strony internetowej zawierającej artykuł serwisu prasowego **www.pudelek.pl** przed wyczyszczeniem i ekstrakcją **metainformacji**.

| |
|---|
| PIĄTEK 04.04.2008<br>Tolak żywi się na bankietach! (BOSKIE ZDJĘCIA)<br>Słowa kluczowe: Jakub Tolak<br>PUDELEK EXCLUSIVE<br>Aktorzy serialowi to wbrew obiegowym opiniom niezamożni ludzie. Jeżeli telenowela jest ich jedynym źródłem dochodów, często żyją na krawędzi, niepewni jutra. Nie sądziliśmy jednak że nie dojadają... |

### 5.1.5. Zapisanie artykułów do bazy wiedzy

Ostatnim etapem indeksowania artykułu jest **writting** tzn. zapisanie go do bazy wiedzy. Polega to na wykonaniu dwóch czynności:

- Zapisaniu artykułu do bazy danych Hipisek,
- Zaindeksowaniu artykułu maszyną indeksującą Sphinx.

Zapisanie artykułu do bazy danych polega na fizycznym zapisaniu wszystkich danych (**metainformacji** i treści artykułu) do bazy SQL. Druga z tych czynności to utworzenie *indeksu wyszukującego* systemu **Sphinx**, który posłuży jako silnik wyszukiwarki tworzonego systemu QA (opis systemu **Sphinx** i jego zastosowania znajduje się w podrozdziale 5.2.1).

## 5.2. Wybór fragmentów tekstu potencjalnie zawierających odpowiedź na analizowane QQuery

**Definicja 5.5** ***Wyszukiwanie odpowiedzi*** *(w systemie **Hipisek.pl**) to proces polegający na przetworzeniu pytania zapisanego w postaci QQuery i zwróceniu kolekcji fragmentów tekstów (bądź artykułów) potencjalnie zawierających odpowiedź na zadane pytanie.*

W podrozdziale przedstawiam, w jaki sposób realizowany jest etap wyszukiwania odpowiedzi w serwisie **Hipisek.pl**. W skrócie omawiam użyty przeze mnie silnik wyszukujący **Sphinx** oraz sposób jego wykorzystania w ramach modułu wyszukującego odpowiedzi projektu **Hipisek.pl** o nazwie **Pythia**.

### 5.2.1. Wyszukiwarka

Silnikiem budowanej wyszukiwarki jest program typu *open-source* o nazwie **Sphinx**[9]. Do podstawowych funkcjonalności tego programu należy:

- Tworzenie indeksu dla zadanej kolekcji dokumentów z wykorzystaniem lematyzacji, listy tzw. *stopwords* (słów które nie będą indeksowane) oraz listy synonimów,

[9]Sphinx free open-source SQL full-text search engine, http://www.sphinxsearch.com

- Znajdowanie identyfikatorów dokumentów spełniających zadane *proste zapytanie boolowskie* (w postaci koniunkcji lub alternatywy słów kluczowych albo fraz) bądź *zapytanie rozszerzone* (w których dodano m.in. operatory *proximity* oraz *quorum matching*[10]),
- Budowanie streszczeń dokumentów w postaci fragmentów zawierających największą ilość spośród podanych słów kluczowych,
- Możliwość pracy w technologii klient-serwer.

Moduł wyszukujący budowanego systemu QA zrealizowałem tworząc z każdego zapytania **QQuery** reprezentującego pytanie, szereg zapytań do wyszukiwarki **Sphinx**. Wyniki zwracane przez **Sphinx'a** poddane zostały integracji i ocenie.

W tym celu zbudowałem za pomocą **Sphinx'a** następujące indeksy na pobranych artykułach:

- Indeks prosty (simple) – indeksuje artykuły nie dokonując na nich żadnych przekształceń,
- Indeks znormalizowany (normalized) – indeksowane artykuły poddawane są normalizacji do postaci samych małych liter,
- Indeks rozszerzony(extended) – najbardziej rozbudowany indeks zawierający w sobie lematyzator (w postaci listy form fleksyjnych), synomizator (w postaci listy synonimów) normalizujący wyrazy do postaci zapisanej za pomocą samych małych liter[11].

Gramatyka zapytań systemu **Sphinx** jest bardzo podobna do zapytań wyszukiwarki *Google*. Podstawowe elementy gramatyki zapytań to:

- Słowa oddzielone spacjami łączone są koniunkcją (każde ze słów musi wystąpić w dopasowanym artykule, niekoniecznie w podanej kolejności),[12]
- Słowa ujęte w cudzysłów traktowane są jako **fraza**, tzn. dokładnie taki ciąg słów musi wystąpić w dopasowanym artykule,
- Poszczególne słowa lub frazy można łączyć alternatywą za pomocą znaku |,
- Do grupowania można stosować nawiasy okrągłe[13].

Wyodrębniłem następujące typy zapytań do Sphinx'a:

- Zapytania budowane w oparciu o frazy wyszukujące QQuery,
- Zapytania typu *„odległość od tematu"*,
- Zapytania typu *„maksymalne wypełnienie słów kluczowych"*.

Zadanie tworzenia zapytań Sphinx'a, oraz oceny zwracanych wyników realizuje program **Pythia**. Wynikiem działania skryptu jest zbiór artykułów, w których potencjalnie znajduje się odpowiedź na **QQuery** reprezentujące pytanie[14].

Poniżej znajduje się opis tworzenia zapytań do **Sphinx'a** z zapytania **QQuery**.

[10]Opis użytych operatorów znajduje się w dalszej części rozdziału.

[11]Lista form fleksyjnych i lista synonimów została wygenerowana ze słownika parsera **Hipisek**.

[12]W praktyce zachowanie to zależy od trybu pracy maszyny **Sphinx**.

[13]W stosowanej w systemie **Hipisek.pl** wersji maszyny **Sphinx** nie można jednak zagnieżdżać nawiasów okrągłych.

[14]Nieprzypadkowo użyłem tu takiego sformułowania. Pamiętać należy, że utworzone zapytanie QQuery może nie oddawać w pełni potrzeby informacyjnej, którą oddawało źródłowe pytanie użytkownika

### 5.2.2. Zapytania budowane w oparciu o frazy wyszukujące QQuery

Najprostszym sposobem wyszukiwania odpowiedzi w systemie **Hipisek.pl** jest metoda oparta na **frazach wyszukujących**, powstałych w procesie analizy pytania i zdefiniowanych w **szablonie pytań** (patrz paragraf 4.1.5). Zapytania takie są z powodzeniem używane np. w systemie opisanym w pracy [28]. Główna idea zapytań w oparciu o frazę wyszukującą systemu **Hipisek.pl** opiera się na następującym założeniu:

**Założenie 5.2** ***Odpowiedź na pytanie*** *znajduje się często w pobliżu pewnej charakterystycznej frazy, którą można wyekstrahować z pytania.*

Dla przykładu rozpatrzmy pytanie:

> Gdzie przebywa teraz Lech Kaczyński?

Dla rozpatrywanego pytania można zbudować szereg fraz wyszukujących postaci:

- Lech Kaczyński przebywa teraz w ...
- Lech Kaczyński właśnie wyjechał do ...
- Lech Kaczyński jest obecnie w ...

Można z dużym prawdopodobieństwem przyjąć, że jeżeli w **bazie wiedzy** znajduje się jedna z powyższych fraz, to następująca po niej partia tekstu będzie odpowiedzią na powyższe pytanie.

Frazy wyszukujące budowane są na etapie zrozumienia pytania. W etapie wyszukania informacji pozostaje więc jedynie wyszukać artykuły zawierające frazy zdefiniowane w ramach danego QQuery. Skrypt **Pythia** może je przesłać bezpośrednio do **Sphinx'a** za pomocą prostego algorytmu:

**Algorytm 5.2** ***Algorytm wyszukiwania odpowiedzi za pomocą fraz wyszukujących***

1. *Niech dany będzie zbiór $R = \emptyset$ oraz zapytanie QQuery $Q$*
2. *Dla każdej z fraz wyszukujących $f \in P_Q$ (gdzie $P_Q$ jest zbiorem wszystkich fraz wyszukujących $Q$) wykonaj następujące kroki:*
   (a) *Utwórz zapytanie $s$ do **Sphinx'a** poprzez otoczenie $f$ cudzysłowem ($s =$ "$f$")*
   (b) *Wyślij $s$ do **Sphinx'a** korzystając z indeksu **simple***
   (c) *Dołóż zbiór wyników $R_s$ zwrócony przez **Sphinx'a** dla zapytania $s$ do zbioru $R$*

Komentarza wymaga tu użycie indeksu **simple**. Na tym etapie wyszukiwania chcemy, aby frazy dawały maksymalnie **adekwatne wyniki**. Użycie indeksu zawierającego np. **lematyzację** spowodowałoby, że fraza *Lech Kaczyński przebywa w* byłaby dopasowana także do tekstu postaci *Lech Kaczyński przebywał w*, który nie musi już prowadzić do dobrej odpowiedzi na rozpatrywane pytanie. Stąd używamy prostszego ze zdefiniowanych indeksów, który wyszukuje frazy dokładnie tak, jak zostały podane.

Niestety pojawia się tu problem związany z obsługą napisów zapisanych w różnym formacie dużych lub małych liter. Użycie indeksu **simple** powoduje, że np. wyszukiwanie frazy *"prezydent Lech Kaczyński znajduje się w"*, w tekście o następującej treści:

> Rozpoczyna się kolejny szczyt Unii Europejskiej. Prezydent Lech Kaczyński znajduje się w Brukseli, gdzie czeka na rozpoczęcie części oficjalnej szczytu.

nie powiedzie się, z powodu niezgodności wielkości pierwszej litery wyrazu: **prezydent**. Aby wyeliminować tego rodzaju problemy wyszukiwanie fraz uruchamiane jest dwukrotnie, za drugim razem z użyciem indeksu **normalized**. Ponieważ jednak może spowodować to powstanie gorszych wyników, ocena artykułów znalezionych w drugim uruchomieniu jest mnożona o stały współczynnik $f$. Wartość współczynnika została dobrana eksperymentalnie i ustawiona na $f = 0.5$ (co można interpretować jako dokumenty o połowę gorsze).

Wynikowy zbiór $R$ jest zbiorem identyfikatorów artykułów zawierających podane frazy wyszukujące i jest pierwszym wynikiem zwracanym przez ten etap przetwarzania pytania.

### 5.2.3. Zapytania typu *„odległość od tematu"*

Druga skonstruowana przeze mnie metoda wyszukiwania odpowiedzi bazuje na następującym założeniu:

**Założenie 5.3** *Jeśli fragment tekstu zawiera odpowiedź na przetwarzane QQuery, to występuje w nim temat pytania oraz akcja pytania. Co więcej, akcja i temat występują blisko siebie. Dokument zawiera również prawie wszystkie słowa wchodzące w skład ograniczeń.*

Przypuśćmy, że mamy pytanie, którego tematem jest wyraz *koncert*, akcją wyraz *odbyć się* natomiast ograniczeniem wyraz *Poznań* (pytaniem takim mogłoby być np. pytanie: *Kiedy odbędzie się koncert w Poznaniu?*). Powyższe założenie oznacza, że jeśli dany tekst zawiera odpowiedź, to najprawdopodobniej:

- Zawiera jedną z fraz: *koncert odbędzie się*, *odbędzie się koncert* , ... ewentualnie z dodaną niewielką ilością wyrazów (np. *odbędzie się dzisiaj koncert*),
- Zawiera co najmniej jedno wystąpienie słowa *Poznań*.

System **Hipisek.pl** korzystając z tego założenia wyszukuje artykuły według podanych w założeniu kryteriów, tzn.:

1. W artykule musi pojawić się **temat pytania**,
2. W artykule musi pojawić się **akcja pytania** w niewielkiej odległości od **tematu**,
3. W artykule musi pojawić się większość ograniczeń.

System skorzysta z maszyny wyszukującej **Sphinx** tworząc odpowiednie zapytanie. W tym wypadku jest ono jednak znacznie bardziej skomplikowane niż w przypadku metody fraz wyszukujących. Tworzone zapytanie składa się z trzech podzapytań połączonych koniunkcją i odpowiadającym kryteriom wymienionym powyżej. System korzysta z indeksu **extended**, aby maksymalnie uprościć tworzone zapytania. Omówię teraz sposób tworzenia każdego z tych trzech podzapytań.

### Podzapytanie dotyczące pojawienia się tematu

Jest to pierwsze z tworzonych podzapytań, składające się z formy bazowej tematu pytania oraz ewentualnych synonimów. W tym celu musimy utworzyć zbiór tekstowych reprezentacji tematu pytania (czyli zbiór napisów składających się ze sklejonych form bazowych, bądź form bazowych synonimów kolejnych leksemów wchodzących w skład tematu pytania). Zadanie to realizuje poniższy algorytm:

**Algorytm 5.3** *Algorytm tworzenia zbioru reprezentacji*

1. *Niech $L$ oznacza ciąg leksemów wchodzących w skład tematu pytania, natomiast $R$ zbiór jednoelementowy zawierający napis pusty.*
2. *Jeśli przetworzono już wszystkie leksemy z $L$, to STOP.*
3. *Pobierz pierwszy leksem $x$ z $L$ i usuń go z $L$.*
4. *Utwórz zbiór $R_x$ składający się z napisów formy bazowej $x$ oraz form bazowych wszystkich synonimów $x$, oraz tymczasowy pusty zbiór napisów $R_t$.*
5. *Dla każdego napisu $r$ ze zbioru $R$ wykonaj:*
   (a) *Dla każdego napisu $s$ ze zbioru $R_x$ wykonaj:*
      i. *Utwórz nowy napis $s_r$ poprzez połączenie spacją napisów $r$ oraz $s$.*
      ii. *Dołóż napis $s_r$ do zbioru $R_t$.*
6. *Przypisz $R := R_t$.*
7. *Przejdź do (2).*

Zbiór $R$ utworzony za pomocą algorytmu będzie wykorzystywany w tworzeniu kolejnych podzapytań i jest **zbiorem reprezentacji tematu**.

Aby utworzyć podzapytanie dotyczące pojawienia się tematu wystarczy teraz każdą z reprezentacji zbioru $R$ otoczyć cudzysłowem, a następnie wszystkie reprezentacje skleić w jeden napis łącząc je znakiem alternatywy $|$.

Przykładowo jeśli **temat pytania** składałby się z dwóch leksemów: *koncert* (którego synonimami są: *występ* i *przedstawienie*) oraz *Chopin* wówczas podzapytanie mogłoby wyglądać w następujący sposób:

```
"koncert Chopin" | "występ Chopin" | "przedstawienie Chopin"
```

Tak skonstruowane podzapytanie dopasuje tylko te artykuły, w których znajduje się co najmniej jedna z podanych fraz.

### Podzapytanie dotyczące pojawienia się akcji

Podzapytanie sprawdzające, czy w tekście pojawiła się akcja w odległości niewielkiej od tematu tworzymy w oparciu o **zbiór reprezentacji tematu pytania** z wykorzystaniem operatora *proximity* systemu **Sphinx**.

Operator *proximity* jest operatorem infiksowym, którego lewym argumentem jest fraza, a prawym liczba całkowita. Operator pozwala na wyszukanie fraz, w których wyrazy wchodzące w skład frazy mogą być oddalone od siebie o zadaną wartość całkowitą. Przykładowo dla zapytania:[15]

[15]Przykład za [29], http://www.sphinxsearch.com

```
"example program"~5
```

Zapytanie dopasuje się do tekstu: *example PHP program*, natomiast nie dopasuje się do tekstu: *example script to introduce outside data into the correct context for your program*, ponieważ dwa z wyrazów są odległe o ponad pięć wyrazów.

Proces tworzenia podzapytania dotyczącego pojawienia się akcji rozpoczyna się od stworzenia zbioru reprezentacji akcji, za pomocą algorytmu 5.3. Oznaczmy tak utworzony zbiór przez $A$. Teraz należy utworzyć frazy podzapytania. Odbywa się to za pomocą następującego algorytmu:

**Algorytm 5.4** ***Algorytm tworzenia podzapytania dotyczącego pojawienia się akcji***

1. *Niech dane będą: zbiór $R$ będący zbiorem reprezentacji tematu pytania oraz zbiór $A$ będący zbiorem reprezentacji akcji (utworzony analogicznie do zbioru $R$ za pomocą algorytmu 5.3).*
2. *Dla każdej reprezentacji $r$ ze zbioru $R$ wykonaj:*
   (a) *Dla każdej reprezentacji $a$ ze zbioru $A$ wykonaj:*
      i. *Utwórz frazę $r_a$ poprzez połączenie spacją fraz $r$ oraz $a$ i otoczeniu cudzysłowem.*
      ii. *Utwórz frazę $a_r$ poprzez połączenie spacją fraz $a$ oraz $r$ i otoczeniu cudzysłowem.*
      iii. *Oblicz parametr proximity $p$ ze wzoru $p = length(r_a) * f$, gdzie $length(x)$ oznacza funkcję zwracającą ilość wyrazów, z których składa się fraza $x$, natomiast $f$ to stały współczynnik.*
      iv. *Utwórz frazy wynikowe $w_{ra}$ oraz $w_{ar}$ poprzez dodanie operatora proximity z argumentem $p$ odpowiednio do $r_a$ oraz $a_r$.*
      v. *Dodaj frazy $w_{ra}$ i $w_{ar}$ do zbioru wyników $W$.*
3. *Połącz frazy ze zbioru wyników $W$ za pomocą alternatywy tworząc gotowe podzapytanie.*

Algorytm uwzględnia sytuację wystąpienia akcji pytania zarówno przed jak i za tematem pytania. Parametr $f$ został dobrany eksperymentalnie i ustawiony na wartość 4.

Przykładowo dla pytania o temacie *koncert* (i synonimie *występ*) oraz akcji *odbyć* (synonim *mieć miejsce*) utworzonoby następujące podzapytanie:

```
"koncert odbyć"~8 | "odbyć koncert"~8 |
"występ odbyć"~8  | "odbyć występ"~8 |
"koncert mieć miejsce"~12 | "mieć miejsce koncert"~12 |
"występ mieć miejsce"~12  | "mieć miejsce występ"~12
```

### Podzapytanie dotyczące ograniczeń

Ostatnim elementem tworzonego zapytania typu *„odległość od tematu"* jest podzapytanie wyszukujące ograniczenia. W celu jego stworzenia skorzystałem z operatora *quorum matching*. Operator ten jest operatorem infiksowym, którego lewym argumentem jest lista słów kluczowych, a prawym liczba całkowita oznaczająca minimalną liczbę wystąpień podanych słów. Przykładowym zapytaniem z wykorzystaniem operatora *quorum matching* jest:[16]

```
"the world is a wonderful place"/3
```

[16]Przykład za [29], http://www.sphinxsearch.com

Zapytanie dopasuje się do tekstów zawierających co najmniej 3 spośród słów: the, world, is, a, wonderful, place.

Gotowe podzapytanie system tworzy scalając formy bazowe wszystkich ograniczeń (tym razem bez synonimów) w jedną frazę, do której stosowany jest operator *quorum matching*. Drugi parametr operatora jest wyliczany poprzez przemnożenie liczby wyrazów tworzących frazę przez stały współczynnik $f = 0.6$. Współczynnik ten dobrałem metodą eksperymentu.

Przykładowo dla zbioru ograniczeń składających się ze słów: *Poznań*, *dzisiaj*, *podróż* utworzone zostanie podzapytanie postaci:

```
"Poznań dzisiaj podróż"/2
```

**Wynikowe zapytanie**

Wynikowe zapytanie powstaje poprzez połączenie koniunkcją trzech stworzonych podzapytań. Następnie trafia ono do maszyny **Sphinx**.

### 5.2.4. Zapytania typu *„maksymalne wypełnienie słów kluczowych"*

Ostatnią metodą wyszukiwania odpowiedzi jest brutalna metoda wyszukująca za pomocą słów kluczowych. Słowami kluczowymi stają się wszystkie formy bazowe tematu, akcji i ograniczeń QQuery (tym razem tworząc zapytanie nie bierzemy pod uwagę synonimów). Zapytanie składa się z listy wymienionych po kolei słów kluczowych, na których zadziałano operatorem *quorum matching*. Parametr operatora zostaje po raz kolejny wyliczony poprzez przemnożenie ilości słów kluczowych przez stały współczynnik (dobrany eksperymentalnie). W metodzie tej korzystamy z indeksu **extended**.

Dodatkowym ograniczeniem zapytania jest wymóg, aby w dokumencie zawsze pojawił się temat pytania. System korzysta tu z podzapytania opisanego w paragrafie 5.2.3.

Metoda ta daje najgorsze wyniki i praktycznie jest wykorzystywana tylko wtedy, jeśli dwie pozostałe metody zawiodą.

## 5.3. Ocena uzyskanych informacji i ich przydatności

Maszyna wyszukująca **Sphinx**, oprócz zwracanych wyników potrafi przyporządkować każdemu z nich wagę (ocenę). Zgodnie z dokumentacją techniczną systemu zawartą w [29] ważenie zwróconych dokumentów może przebiegać na różne sposoby. Są to między innymi:

1. Brak ważenia (wszystkie zaakceptowane dokumenty otrzymują stałą wagę równą 1),
2. **Ranking statystyczny** – ranking wyliczany jest za pomocą funkcji *Okapi BM25* uwzględniającej frekwencję dopasowanych wyrazów w całej kolekcji dokumentów[17],
3. **Ranking frazowy** – ranking wyliczany w oparciu o najdłuższy wspólny podciąg pomiędzy dopasowanym dokumentem, a zapytaniem.

Wszystkie zapytania kierowane do maszyny **Sphinx** uruchamiane są w trybie, w którym końcowa waga wynikowych dokumentów powstaje poprzez zastosowanie metod

[17]Dokładny opis działania funkcji BM25 można znaleźć w pracy [5].

oceny (2) oraz (3) i obliczeniu ich **sumy ważonej** przemnożeniu przez 1000 i sprowadzeniu do liczby całkowitej. Taka ocena zwracana jest wraz z dokumentem przez maszynę **Sphinx**.

W systemie **Hipisek.pl** wykorzystujemy oceny podane przez **Sphinx'a** do wyprodukowania ocen dokumentów wyszukiwanych w omawianym etapie wyszukiwania odpowiedzi. Oceny końcowe powstają poprzez przeskalowanie ocen maszyny **Sphinx** na zbiór $[0, 1]$. Robimy to za pomocą następującego algorytmu:

**Algorytm 5.5** ***Algorytm oceny wyszukanych odpowiedzi***

1. *Niech dany będzie zbiór identyfikatorów wyszukanych dokumentów $D_x$ przez metodę $x$ oraz zbiór ocen metody $x$ wyszukiwania odpowiedzi $R_x = \{r_i : i \in D_x\}$, gdzie $r_i$ oznacza ocenę wyszukanego dokumentu o identyfikatorze $i$*
2. *Policz $max_R = \max(R_x)$ oraz $min_R = \min(R_x)$*
3. *Znormalizuj wagę każdego z dokumentów o identyfikatorach ze zbioru $D_x$ poprzez zastosowanie wzoru: $newr_i = \frac{max_R - r_i}{max_R - min_R}$*
4. *Przypisz każdemu z dokumentów o identyfikatorach ze zbioru $D_x$ nową wagę obliczoną w poprzednim punkcie algorytmu*

Warto zauważyć, że algorytm normalizuje wagi tylko w ramach danej metody wyszukania, przez co np. oceny metody frazowej i słów kluczowych są nadal nieporównywalne między sobą. Aby usunąć ten problem przyjąłem, że ocena musi być pomnożona przez pewien współczynnik $f_x \in (0, 1]$ stały dla danej metody $x$. Im dana metoda *daje lepsze wyniki* tym wyższy jest jej współczynnik $f_x$. Wartości współczynnika zostały dobrane w drodze eksperymentów i ustawione na:

- Dla metody frazowej: $f_x = 1$
- Dla metody odległości od tematu: $f_x = 0.4$
- Dla metody wypełnienia słów kluczowych: $f_x = 0.05$

Metoda frazowa została uznana przeze mnie za najlepszą i dającą potencjalnie najlepsze wyniki, natomiast metoda odległości od tematu jest znacznie lepsza od metody wypełnienia słów kluczowych, stąd taki rozkład współczynników $f_x$.

ROZDZIAŁ 6

# Ekstrakcja odpowiedzi

W rozdziale omawiam sposób **ekstrakcji odpowiedzi** z wyszukanych artykułów znalezionych w etapie **wyszukania odpowiedzi**. Rozdział rozpoczyna krótki opis podstawowych zadań tego etapu przetwarzania pytania. Następnie omawiam metody znajdowania kandydatów odpowiedzi. Rozdział kończy opis oceny kandydatów odpowiedzi oraz sposobu ich akceptacji do wynikowego zbioru odpowiedzi.

## 6.1. Zdefiniowanie zadania ekstrakcji odpowiedzi

Ekstrakcja odpowiedzi jest ostatnim etapem przetwarzania pytania w systemie **Hipisek.pl**. Jej podstawowym zadaniem jest znalezienie w wyszukanych dokumentach fragmentu tekstu, który spełnia definicję odpowiedzi 3.9. Przyjąłem założenie, że odpowiedzią jest pojedyncze zdanie znalezione w wyszukanych dokumentach. Zdanie takie **nie zostaje** poddane żadnym modyfikacjom i trafia do użytkownika w formie, w jakiej wystąpiło w tekście źródłowym.

Oprócz odpowiedzi użytkownik otrzymuje szereg dodatkowych informacji, w postaci:

- **Kontekstu odpowiedzi** w postaci zdania poprzedzającego odpowiedź i zdania następującego po odpowiedzi (jeśli zdania takie istnieją),[1]
- Adresu URL źródłowego artykułu,
- Daty opublikowania artykułu.

System **Hipisek.pl** nie udostępnia tylko jednej odpowiedzi. W przypadku znalezienia większej ich ilości użytkownik otrzymuje **listę odpowiedzi** posortowaną malejąco według **ocen**.

Podsumowując, w etapie **ekstrakcji odpowiedzi** możemy wyróżnić następujące podetapy:

- Znalezienie kandydatów odpowiedzi,
- Ocena znalezionych kandydatów odpowiedzi i ich weryfikacja,
- Podanie odpowiedzi użytkownikowi.

W kolejnych podrozdziałach omówię dwa pierwsze z wyróżnionych podetapów. Podanie odpowiedzi, jako proces czysto techniczny nie będzie omówiony w ramach niniejszego rozdziału.

[1] Ma to na celu zwiększenie skuteczności serwisu. Faktyczna odpowiedź może mieć sens tylko w ramach danego kontekstu.

## 6.2. Znalezienie kandydatów odpowiedzi

Proces znalezienia kandydatów odpowiedzi można zapisać w postaci następującego algorytmu:

**Algorytm 6.1** *Algorytm znajdowania kandydatów odpowiedzi*

1. *Niech dany będzie zbiór $D$ identyfikatorów artykułów wyszukanych w procesie wyszukiwania odpowiedzi, zapytanie QQuery $Q$ oraz pusty zbiór znalezionych kandydatów $C$.*
2. *Jeśli zbiór $D$ jest pusty to STOP.*
3. *Ze zbioru $D$ pobierz porcję $D_{part}$ artykułów o najwyższej ocenie, usuwając je ze zbioru $D$.*
4. *Dla każdego artykułu $d$ ze zbioru $D_{part}$ wykonaj:*
   (a) *Jeśli $d$ wyszukano z wykorzystaniem **fraz wyszukujących** rozpocznij znajdowanie kandydatów za pomocą **metody frazy wyszukującej**.[2] Zapisz znalezionych kandydatów do zbioru $C$. Następnie przejdź do punktu (4b).*
   (b) *Znajdź odpowiedzi za pomocą **metody wystąpienia tematu**.[3] Zapisz znalezionych kandydatów do zbioru $C$.*
   (c) *Jeżeli zbiór $C$ zawiera więcej odpowiedzi niż $min_{answers}$, to STOP.*
   (d) *W przeciwnym przypadku przejdź do punktu (2).*

Parametr $min_{answers}$ oznacza minimalną liczbę kandydatów odpowiedzi jaką powinien znaleźć serwis **Hipisek.pl**.

Podczas znajdowania odpowiedzi przyjmuję następujące założenie:

**Założenie 6.1** *Zdanie, będące odpowiedzią na pytanie znajduje się w treści wyszukanych artykułów.*

Dzięki temu założeniu w procesie znajdowania odpowiedzi, kolejno odrzucam zdania, które na pewno nie mogą być odpowiedzią. Pozostałe zdania trafiają do **zbioru kandydatów odpowiedzi**. Do utworzenia tego zbioru wykorzystuję dwie opracowane przeze mnie metody:

- Metodę frazy wyszukującej,
- Metodę wystąpienia tematu.

W kolejnych paragrafach omawiam wyżej wymienione metody.

### 6.2.1. Metoda frazy wyszukującej

W metodzie po raz kolejny wykorzystywane są **frazy wyszukujące** utworzone za pomocą **szablonu pytania**. Korzystamy raz jeszcze z założenia 5.2, według którego odpowiedź na pytanie znajduje się w pobliżu frazy wyszukującej. Przyjmiemy, że wystąpienie takiej frazy jest wystarczającym warunkiem, aby zdanie artykułu było dobrym **kandydatem odpowiedzi**.

Schemat metody fraz wyszukujących można zapisać za pomocą prostego algorytmu:

[2] Opis znajdowania kandydatów odpowiedzi za pomocą metody frazowej znajduje się Podrozdziale 6.2.1.

[3] Opis znajdowania kandydatów odpowiedzi za pomocą metody wystąpienia tematu znajduje się Podrozdziale 6.2.2.

**Algorytm 6.2** ***Algorytm znajdowania kandydatów odpowiedzi metodą frazy wyszukującej***

1. *Niech dany będzie identyfikator $d_x$ artykułu wyszukanego w procesie wyszukiwania odpowiedzi za pomocą metody frazy wyszukującej, zapytanie QQuery $Q$ oraz pusty zbiór znalezionych kandydatów $C_p$.*
2. *Utwórz wyrażenie regularne $r_p$ składające się z alternatywy wszystkich fraz wyszukujących zapytania $Q$.*
3. *Dla każdego zdania $s$ artykułu $d_x$ wykonaj:*
   (a) *Jeśli $s$ dopasowuje się do $r_p$, to dodaj $s$ do zbioru kandydatów z oceną początkową równą $p$.*
   (b) *W przeciwnym przypadku sprawdź, czy $s$ dopasowuje się do $r_p$ ignorując wielkość liter. Jeśli tak, to dodaj $s$ do zbioru kandydatów z oceną początkową równą $p_{ignore\_case}$.*

**Ocena początkowa** odpowiedzi jest wyjściowym współczynnikiem dla końcowej oceny odpowiedzi. Celem jej wprowadzenia jest rozróżnienie pomiędzy kandydatami znalezionymi za pomocą różnych metod, lub znalezionych w różny sposób w ramach danej metody. Parametry ocen początkowych metody fraz wyszukujących mają następującą interpretację:

- $p$ – ocena początkowa w przypadku znalezienia frazy dokładnie tak jak została zapisana,
- $p_{ignore\_case}$ – ocena początkowa w przypadku znalezienia frazy znormalizowanej, przy czym $p > p_{ignore\_case}$.

Przyjąłem następujące wartości ocen początkowych:

- $p = 1$
- $p_{ignore\_case} = 0.8$

### 6.2.2. Metoda wystąpienia tematu

Druga opracowana przeze mnie metoda szuka zdań o następujących własnościach:

- Zdanie zawiera **temat pytania**,
- Zdanie wraz z kontekstem zawiera **akcję pytania**,
- W zdaniu wraz kontekstem występuje *większość* **ograniczeń** pytania.

Zadanie to realizowane jest za pomocą algorytmu:

**Algorytm 6.3** ***Algorytm znajdowania kandydatów odpowiedzi metodą wystąpienia tematu***

1. *Niech dany będzie identyfikator $d_x$ artykułu wyszukanego w procesie wyszukiwania odpowiedzi, zapytanie QQuery $Q$ oraz pusty zbiór znalezionych kandydatów $C_t$.*
2. *Utwórz wyrażenie regularne $r_t$ poprzez sklejenie alternatywą napisów wszystkich pól **tematu pytania** spośród wymienionych: użyta forma, forma bazowa, dowolna forma fleksyjna, dowolna forma fleksyjna dowolnego z synonimów.*

3. *Utwórz zbiór $S_t$ zdań dokumentu $d_x$, które dopasowują się do wyrażenia regularnego $r_t$ oraz zbiór $S_{ti}$ zdań dokumentu $d_x$, które nie dopasowują się do wyrażenia regularnego $r_t$, ale dopasowują się do wyrażenia $r_t$ przy zignorowaniu wielkości liter.*

4. *Jeśli zbiory $S_t$ oraz $S_{ti}$ są puste, to STOP.*

5. *Utwórz wyrażenie regularne $r_a$ analogicznie do $r_t$ wykorzystując **akcję pytania**.*

6. *Ze zbiorów $S_t$ oraz $S_{ti}$ usuń zdania, które wraz z kontekstem nie dopasowują się do wyrażenia $r_a$ przy zignorowaniu wielkości liter.*

7. *Jeśli zbiory $S_t$ oraz $S_{ti}$ są puste, to STOP.*

8. *Dla każdego leksemu $c_i$ ze **zbioru ograniczeń** utwórz wyrażenie regularne $r_{ci}$ analogicznie do $r_t$.*

9. *Dla każdego zdania $s$ ze zbiorów $S_t$ oraz $S_{ti}$ oblicz wskaźnik spełnienia ograniczeń $f_c(s)$ w następujący sposób:*

   (a) *Oblicz $constraint_{match}$ – ilość wyrażeń regularnych $r_{ci}$, do których dopasowuje się zdanie $s$ wraz z kontekstem.*

   (b) *Oblicz $constraint_{match\_ic}$ – ilość wyrażeń regularnych $r_{ci}$, które nie dopasowały się w kroku (9a), a do których dopasowuje się zdanie $s$ wraz z kontekstem przy zignorowaniu wielkości liter.*

   (c) *Przypisz $f_c(s)$ korzystając ze wzoru:*

$$f_c(s) = \begin{cases} \frac{constraint_{match} + constraint_{ic} * constraint_{match\_ic}}{count_{constraint}} & \text{dla } count_{constraint} > 0 \\ 1 & \text{w p.p.} \end{cases}$$

   *gdzie $count_{constraint}$ oznacza ilość ograniczeń QQuery.*

10. *Ze zbiorów $S_t$ oraz $S_{ti}$ usuń zdania $s$, których wskaźnik spełnienia ograniczeń $f_c(s)$ nie przekracza parametru $constraint_{accept}$.*

11. *Dodaj do wynikowego zbioru kandydatów $C_t$ zdania ze zbiorów $S_t$ oraz $S_{ti}$ z oceną początkową $p$ równą:*

    (a) *Jeśli zdanie $s$ pochodzi ze zbioru $S_t$, to $p = topic_{factor} * f_c(s)$.*

    (b) *Jeśli zdanie $s$ pochodzi ze zbioru $S_{ti}$, to $p = topic_{factor\_ic} * f_c(s)$.*

Współczynniki algorytmu: $constraint_{ic}$, $constraint_{accept}$, $topic_{factor}$ oraz $topic_{factor\_ic}$ są stałe i zostały dobrane heurystycznie. Przyjąłem następujące wartości współczynników:

- $constraint_{ic} = 0.9$
- $constraint_{accept} = 0.55$
- $topic_{factor} = 0.75$
- $topic_{factor\_ic} = 0.6$

## 6.3. Ranking odpowiedzi

Ranking odpowiedzi ma na celu:

1. Usunięcie ze zbiorów kandydatów zdań, które nie są dobrymi odpowiedziami na zadane pytanie.
2. Ocenę jakości odpowiedzi ze zbioru kandydatów.

Do realizacji tych zadań wprowadziłem następujące kategorie ocen kandydatów:

1. **Ocena konieczna** – jest to ocena weryfikująca, czy kandydat $c$ powinien trafić do wynikowego zbioru odpowiedzi, podanych przez system.
2. **Ocena wystarczająca** – jest to ocena, która wraz z **oceną konieczną** decyduje o jakości odpowiedzi. Ocena wystarczająca nie ma wpływu na umieszczenie kandydata do wynikowego zbioru odpowiedzi.
3. **Ocena wynikowa** – jest to ocena gotowej odpowiedzi podawanej przez system. Jest wyliczana z oceny koniecznej, oceny wystarczającej oraz początkowej oceny obliczonej na etapie znajdowania odpowiedzi (patrz Podrozdziały 6.2.1 i 6.2.2).

Oceny te są **liczbami rzeczywistymi**. W kolejnych paragrafach omawiam sposób obliczenia **oceny koniecznej**, **oceny wystarczającej** oraz wyliczanej z nich **wynikowej oceny odpowiedzi**.

### 6.3.1. Oceny konieczne

**Ocena konieczna** ($r_k$) kandydata odpowiedzi $s$ obliczona jest jako suma ważona wektora, którego kolejne elementy przyjmują następujące wartości (w sumowaniu nie biorą udziału elementy niezdefiniowane):

1. Parametr $f_{tt}$ spełnienia podtypu „typ odpowiedzi" $t_t$ zapytania QQuery, wyliczany w następujący sposób:[4]
   - Jeśli $t_t = UNKNOWN$, to $f_{tt}$ ma wartość niezdefiniowaną.
   - Jeśli $t_t = PLACE$, to $f_{tt}$ jest równe: 1, jeśli w zdaniu $s$ wraz z kontekstem występują wyrażenia związane z miejscem i przestrzenią, natomiast 0 w p.p.
   - Jeśli $t_t = TIME$, to $f_{tt}$ jest równe: 1, jeśli w zdaniu $s$ wraz z kontekstem występują wyrażenia związane z czasem, natomiast 0 w p.p.
2. Parametr $f_{ts}$ spełnienia podtypu „charakter wymaganej wiedzy" QQuery $t_s$, wyliczany w następujący sposób:
   - Jeśli $t_s = UNKNOWN$ lub $t_s = FUTURE$[5], to $f_{ts}$ ma wartość niezdefiniowaną.
   - Jeśli $t_s = PAST$, to $f_{ts}$ przypisz wartość $w$ ze zbioru $[0, 1]$, zależną od daty opublikowania artykułu źródłowego odpowiedzi. Wartość $w$ jest tym większa im artykuł źródłowy jest starszy.

[4]Sprawdzenie warunków przy obliczaniu elementu (1) wektora następuje z wykorzystaniem wyrażeń regularnych oraz **parsera Hipisek**. Opis parsera Hipisek znajduje się w Podrozdziale 4.1.2.

[5]Oznacza to, że *de facto* oznaczenie typu odpowiedzi jako FUTURE nie jest wykorzystywane w ramach systemu Hipisek. Typ ten został wprowadzony dla pełności prezentowanej taksonomii pytań i stanowi jeden z elementów rozwinięcia projektu (patrz Podrozdział 7.3).

- Jeśli $t_s = PRESENT$, to $f_{ts}$ przypisz wartość $w$ ze zbioru $[0, 1]$, zależną od daty opublikowania artykułu źródłowego odpowiedzi. Wartość $w$ jest tym większa im artykuł źródłowy jest młodszy.

3. Czy wystąpiła fraza wyszukująca lub czy zbiór fraz wyszukujących QQuery jest pusty? (1 jeśli warunek jest spełniony, 0 w p.p.).

4. Stosunek ograniczeń występujących w zdaniu odpowiedzi wraz z kontekstem do ilości wszystkich ograniczeń QQuery lub 1 jeśli zbiór ograniczeń jest pusty.

Po wyliczeniu **ocen koniecznych** kandydatów odpowiedzi, następuje weryfikacja odpowiedzi. Do zbioru wynikowych odpowiedzi trafiają kandydaci, których **ocena konieczna** $r_k$ przekracza próg akceptujący $f_{accept}$. Próg ten wyliczany jest w oparciu o maksymalną możliwą wartość wektora oceny koniecznej $max(r_k)$ (nie są brane pod uwagę elementy niezdefiniowane). Dla celów dalszych obliczeń **ocena konieczna** jest normalizowana do zbioru $[0, 1]$ poprzez podzielenie przez wartość $max(r_k)$.

☞ **Uwaga** Warto zauważyć, że przy takiej konstrukcji oceny koniecznej, w przypadku gdy dla przetwarzanego zapytania QQuery $q$ zbiory ograniczeń i fraz wyszukujących są puste, oraz typ pytania jest równy (UNKNOWN, UNKNOWN) lub (UNKNOWN, FUTURE), wszyscy kandydaci są automatycznie akceptowani.

### 6.3.2. Oceny wystarczające

Ocena wystarczająca wyliczana jest jako suma ważona wektora o kolejnych elementach wyliczanych w następujący sposób:

1. Współczynnik długości odpowiedzi. Im odpowiedź dłuższa, tym większy współczynnik.

2. Czy artykułowi przypisano słowa kluczowe występujące w którymś z ograniczeń lub w temacie pytania? (1 jeśli warunek jest spełniony, 0 w p.p.).

3. Stały współczynnik jakości źródła (serwisu prasowego) z którego pochodzi artykuł.

Dla celów dalszych obliczeń **ocena wystarczająca** jest normalizowana do zbioru $[0, 1]$ poprzez podzielenie przez maksymalną możliwą do osiągnięcia wartość oceny wystarczającej.

### 6.3.3. Wynikowa ocena odpowiedzi

Wynikowa ocena $r$ odpowiedzi $s$ powstaje poprzez wykorzystanie następujących ocen cząstkowych:

- Oceny początkowej $r_p$,
- Oceny koniecznej $r_k$,
- Oceny wystarczającej $r_w$.

Do wyliczenia oceny $r$ stosuję wzór:

$$r = r_p * (w_k * r_k + w_w * r_w)$$

Gdzie $w_k$ oraz $w_w$ oznaczają wagi ocen koniecznej i wystarczającej. Współczynniki te zostały dobrane metodą heurystyczną. Przyjąłem następujące wartości wag:

- $w_k = 0.75$
- $w_w = 0.25$

Zaakceptowana odpowiedź wraz z oceną $r$ trafia do warstwy prezentacyjnej systemu **Hipisek.pl**. Serwis prezentuje odpowiedzi w kolejności od najwyżej do najniżej ocenionej.

ROZDZIAŁ 7

# Prezentacja systemu i podsumowanie

## 7.1. Przedstawienie serwisu

Serwis **Hipisek.pl** został uruchomiony w październiku 2008 roku. Początkowo służył głównie zebraniu korpusu pytań wyszukiwarki (patrz podrozdział 2.1.2), choć zaimplementowano w nim pewne mechanizmy wyszukujące informacje. Uruchomienie instancji projektu, będącej efektem niniejszej pracy miało miejsce **15 maja 2009 roku**. W pracy omawiam stan systemu z dnia uruchomienia.

### 7.1.1. Użyte technologie

Główne elementy projektu **Hipisek.pl** zostały napisane w języku **Perl**.[1] Perl jest językiem skryptowym, szczególnie nadającym się do analizy i przetwarzania tekstu. W projekcie wielokrotnie wykorzystywano jego sztandarowe funkcjonalności i cechy, takie jak:

- Możliwość łatwego i efektywnego korzystania z **wyrażeń regularnych**,
- Wsparcie dla programów operujących na tekście,
- Łatwość tworzenia programów,
- Bogaty zbiór bibliotek i modułów.[2]

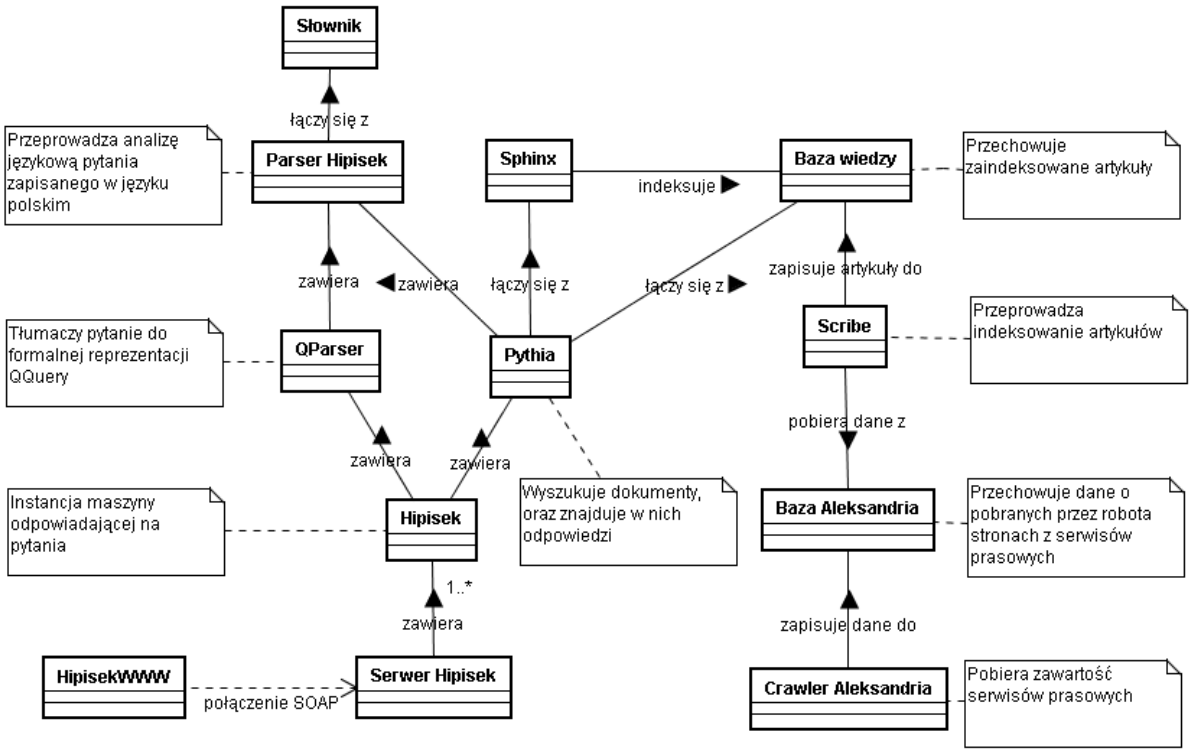
Niestety podstawową wadą języka Perl jest niska szybkość działania. Choć nie jest to tradycyjny język skryptowy (program nie jest wykonywany jak typowy skrypt lecz jest najpierw kompilowany do kodu pośredniego, podobnie jak np. język Java), to szybkość jego działania pozostawia wiele do życzenia. W komercyjnym projekcie jest to aspekt bardzo istotny, przez co wybór języka **Perl** *właściwie* dyskwalifikuje serwis **Hipisek.pl** jako produkt komercyjny.[3]

Wykorzystanym systemem obsługi baz danych jest **PostgreSQL**[4]. Wyboru tego dokonano z uwagi na następujące cechy systemu:

- Jest to oprogramowanie darmowe,
- Udostępnia zaawansowane możliwości obsługi danych (w tym m.in. tworzenie różnych typów indeksów oraz definiowanie funkcji),
- Ma zadowalającą szybkość działania.

System **Hipisek.pl** działa w postaci aplikacji typu *klient-serwer*. **Klientem** jest warstwa prezentacyjna serwisu, umieszczona na serwerze WWW i napisana w PHP. Klient komunikuje się z serwerem za pomocą protokołu SOAP.

[1] Opis języka **Perl** można znaleźć m.in. na stronie internetowej *Perl directory*, http://www.perl.org
[2] Głównym repozytorium modułów języka Perl jest CPAN, http://www.cpan.org
[3] Propozycja zaimplementowania serwisu w szybszym języku opisana została w Rozdziale 7.3.
[4] Strona domowa projektu PostgreSQL, http://www.postgresql.org




Rysunek 7.1: Architektura projektu **Hipisek.pl**

System umieszczony został na serwerach udostępnianych przez **rootnode.net**.[5] Wykorzystano następujące maszyny:

- Serwer ogólnego przeznaczenia **Stallman**, na którym znajdują się bazy projektu **Hipisek** oraz na którym działa **robot Aleksandria**,
- Serwer **Torvalds**, na którym znajduje się uruchomiony serwer **Hipisek.pl** oraz maszyna indeksująca **Sphinx**,
- Serwer **Venema**, obsługujący stronę WWW **http://www.hipisek.pl**.

### 7.1.2. Architektura

Architektura systemu została przedstawiona na diagramie 7.1. Na diagramie wyszczególniono podstawowe moduły z jakich składa się projekt w tym:

- Główny moduł systemu odpowiedzialny za dostarczenie odpowiedzi na zadane pytanie – **Hipisek**,
- Moduł serwera obsługujący żądania użytkowników – **Serwer Hipisek**
- Warstwa prezentacyjna, w postaci strony WWW http://www.hipisek.pl,
- Parser Hipisek,
- Moduł translacji do QQuery – **QParser**,

[5] Rootnode, konta shellowe z piekła, http://www.rootnode.net

- Moduł wyszukujący artykuły i znajdujący odpowiedzi – **Pythia**,
- Pomocniczy moduł obsługujący maszynę indeksującą – **Sphinx**,
- Pomocniczy moduł pobierający strony z serwisów prasowych – **robot Aleksandria**,
- Pomocniczy moduł indeksujący artykuły – **Scribe**.

Na diagramie nie wyszczególniono niektórych stworzonych w ramach projektu modułów pomocniczych, w tym między innymi:

- Skryptów tworzących korpusy pytań oraz zliczających ich statystyki,
- Skryptów tworzenia, konserwacji i zarządzania słownikiem,
- Skryptów testujących system **Hipisek.pl**.

### 7.1.3. Zasoby

#### Słownik

Słownik projektu **Hipisek.pl** powstał poprzez kompilację kilku ogólnodostępnych źródeł[6] oraz ręczną edycję problemowych słów. Statystyki słownika znajdują się w tabeli 7.1.

Tabela 7.1: Statystyka słownika systemu **Hipisek.pl**

| | |
|---|---|
| Ilość leksemów ogółem | 128428 |
| Leksemy z oznaczoną semantyką | 3865 |
| Leksemy z oznaczoną częścią mowy | 97739 |
| Leksemy będące idiomami | 9969 |
| Leksemy będące nazwą własną | 32162 |
| Ilość różnych form fleksyjnych | 590918 |
| Ilość powiązań synonimicznych | 166170 |

#### Baza wiedzy

Programy tworzące bazę wiedzy uruchomione zostały **1 maja 2009 roku**. Indeksowano artykuły pojawiające się w ramach serwisów:

- www.pudelek.pl
- www.tvn24.pl

Przez prawie trzy tygodnie działania zebrano znaczny materiał źródłowy i zapisano w bazie wiedzy. Zaindeksowano **7414** artykułów, co stanowi ponad **14 MB** *litego tekstu*.

[6] Lista użytych źródeł znajduje się w podrozdziale 4.1.2.

## 7.2. Ewaluacja serwisu

### 7.2.1. Opis procesu ewaluacji

Ewaluacja projektu **Hipisek.pl** polegała na porównaniu wyników serwisu z wynikami dostarczonymi przez serwisy google.pl oraz ktoco.pl. Test składał się z **25 pytań** o czas lub miejsce, których odpowiedź powinna potencjalnie znajdować się w bazie wiedzy projektu **Hipisek.pl**. Zestaw pytań ułożyłem w oparciu o przykłady z zebranego korpusu pytań. Pełna lista pytań wchodzących w skład zestawu testującego znajduje się w dodatku D. Testerzy mieli za zadanie przyporządkować każdej odpowiedzi ocenę w skali od 0 do 5.

Kryteria oceny były następujące:

- Dostarczenie odpowiedzi,
- Ilość nieprawidłowych odpowiedzi,
- Dokładność odpowiedzi,
- Adekwatność odpowiedzi do zadanego pytania.

Testerzy nie brali pod uwagę takich czynników jak:

- Estetyka wykonania serwisu,
- Czas działania.

### 7.2.2. Omówienie wyników testu

Test przeprowadzono na grupie **5 osób**. Wyniki testu zawiera tabela 7.2.

Tabela 7.2: Wyniki ewaluacji systemu **Hipisek.pl**

| | Google | Ktoco | Hipisek |
|---|---|---|---|
| Ocen 0 | 20.5% | 36.7% | 14.3% |
| Ocen 1 | 16.7% | 12.7% | 14.3% |
| Ocen 2 | 14.1% | 8.9% | 7.8% |
| Ocen 3 | 12.8% | 8.9% | 13.0% |
| Ocen 4 | 19.2% | 16.5% | 24.7% |
| Ocen 5 | 16.7% | 16.5% | 26.0% |
| Średnia ocena | 2.44 | 2.05 | 2.97 |

W ocenie testerów serwis **Hipisek.pl** dostarczał **najlepsze odpowiedzi** na pytania zawarte w zestawie testowym, wyprzedzając serwis **Google** o 0.53 punktu, oraz analogiczną technologicznie „metawyszukiwarkę semantyczną" **Ktoco.pl** o 0.92 punktu. Wyniki ewaluacji potwierdzają zasadność przyjętego modelu dla zdefiniowanego typu pytań oraz ich dziedziny.

## 7.3. Możliwe rozwinięcia projektu

Projekt **Hipisek.pl** jest pierwszą wersją powstającego serwisu odpowiadającego na pytania. Potencjał rozwoju projektu jest duży i obejmuje rozbudowę bądź wprowadzenie takich elementów jak:

- Obsługa większej ilości pytań,
  - Dodanie obsługi nowych typów pytań, innych niż o czas i miejsce,
  - Obsługa pytań złożonych,
  - Obsługa pytań dotyczących wiedzy encyklopedycznej i ogólnej (np. pytań: *Kiedy była bitwa pod Waterloo?*, *Która godzina?*, *Co to jest dzbanek?*),
  - Dodanie nowych mechanizmów obsługi pytania i wyszukiwania odpowiedzi (np. wnioskowania, ontologii).
- Rozszerzenie bazy wiedzy
  - Indeksowanie dowolnych artykułów sieci internet,
  - Dodanie sprzęgu do innych baz wiedzy, np. DBPedia,[7]
  - Dodanie informacji multimedialnych, np. grafiki lub dźwięku.
- Użycie zaawansowanych narzędzi przetwarzania języka naturalnego,[8]
- Przyspieszenie serwisu
  - Przepisanie kluczowych elementów serwisu na język C++,
  - Zrównoleglenie działania serwisu na więcej maszyn.
- Zaprojektowanie nowoczesnego interfejsu użytkownika
  - Dodanie obsługi zadawania pytań za pomocą mowy,
  - Dodanie możliwości przeczytania odpowiedzi przez syntezator mowy,
  - Wprowadzenie elementów rozmowy i ewolucja serwisu w stronę aplikacji typu *chatter-bot*.[9]

[7] *The DBpedia Knowledge Base* http://dbpedia.org

[8] W trakcie pracy nad serwisem podjąłem próbę użycia parsera napisanego przez Filipa Gralińskiego (patrz [30]) i używanego w systemie tłumaczenia automatycznego Translatica. Niestety z powodów technicznych, pomysł wykorzystania tego narzędzia został przeze mnie tymczasowo zarzucony.

[9] Zadaniem aplikacji typu *chatter-bot* jest symulowanie rozmowy z prawdziwym człowiekiem. Aplikacje tego typu znajdują obecnie zastosowanie, jako *wirtualni asystenci* (np. http://www.chatbot.pl)

# Podsumowanie

W pracy opisałem zbudowany przeze mnie system QA **Hipisek.pl**. Omówiłem podstawowe założenia *systemów odpowiadania na pytania* oraz przedstawiłem przykłady działających systemów komercyjnych i projektów badawczych. Przedstawiłem prace związane z przygotowaniem do budowy autorskiego serwisu polegające na zbieraniu materiału korpusowego. Następnie omówiłem model teoretyczny systemu oraz wprowadziłem formalizm reprezentowania pytania za pomocą struktury QQuery. W kolejnych rozdziałach dokładnie przedstawiłem realizację istotnych elementów modelu poprzez omówienie trzech kluczowych etapów przetwarzania pytania: zrozumienia pytania, wyszukania odpowiedzi oraz ekstrakcji odpowiedzi. Dodatkowo omówiłem sposób pozyskiwania i przechowywania bazy wiedzy w stworzonym systemie. Przedstawiłem ewaluację serwisu w oparciu o zestaw testowych pytań oraz krótko wymieniłem możliwe elementy jego dalszego rozwoju.

Serwis **Hipisek.pl** powstały w wyniku niniejszej pracy dostępny jest pod adresem http://www.hipisek.pl. Rozpoczęto prace nad rozbudową serwisu i rozszerzeniem modelu projektu opisywanego w ramach niniejszej pracy magisterskiej.

DODATEK A

# Gramatyka zapytań QQuery

## A.1. Konwencja zapisu gramatyki pytań

Przyjąłem następujące konwencje nazewnicze:

- Wszystkie nazwy tworzę w języku angielskim, co podyktowane zostało przyjętym standardem kodowania programów
- Literały pisane samymi dużymi literami oznaczają **termy atomowe** (z wyjątkiem literałów zaczynających się od **VALUE** patrz niżej)
- Literały zaczynające się od **VALUE** traktujemy jako literały o określonym typie wartości (np. liczby, tekst etc.)
- Pozostałe literały zapisałem używając notacji typu *CamelCase* (tzn. pierwsza litera każdego z wyrazów literału jest wielka, pozostałe są małymi literami)
- Znak * pojawiający się z prawej strony literału, oznacza listę (także pustą) elementów będących danym literałem
- Jeśli prawa strona reguły zawarta jest w nawiasach okrągłych „()", to reguła oznacza, że literał występujący po jej lewej stronie jest listą (uporządkowaną) wymienionych w nawiasach literałów
- Jeśli prawa strona reguły zawarta jest w nawiasach klamrowych „{}", to reguła oznacza, że literał występujący po jej lewej stronie przyjmuje jedną z wartości występujących w nawiasach
- Czcionką pogrubioną oznaczono komentarze

## A.2. Typy wartości termów atomowych

Termy atomowe mogą przyjmować wartości jednego z typów:

- **VALUE_TEXT** – wartość tekstowa
- **VALUE_INTEGER** – liczba całkowita
- **VALUE_DOUBLE** – liczba zmiennoprzecinkowa (rzeczywista)

Literały zapisane dużymi literami traktujemy jako **stałe wartości tekstowe**.

## A.3. Gramatyka

**Cała struktura QQuery:**
QQuery:- (Type, Topic, Action, Constraints, Phrases)
**Typ QQuery:**
Type:- (AnswerType, TimeStateType)
AnswerType:- {TIME, PLACE, UNKNOWN}
TimeStateType:- {PAST, PRESENT, FUTURE, UNKNOWN}
**Temat QQuery:**
Topic:- (LexemeWithRank*)
**Akcja QQuery:**
Action:- (LexemeWithRank*)
**Zbiór ograniczeń:**
Constraints:- (LexemeWithRank*)
**Leksem wraz z oceną:**
LexemeWithRank:- (Lexeme,Rank)
**Zbiór fraz wyszukujących:**
Phrases:- (Phrase*)
Phrase:- ( PhraseText, Rank )
**Typy danych:**
**Leksem:**
Lexeme:- (UsedForm, Canon, AllForms, Synonyms, Attribs, NormalizedForm)
**Forma w jakiej wystąpił leksem:**
UsedForm:- {VALUE_STRING}
**Forma kanoniczna leksemu:**
Canon:- {VALUE_STRING}
**Zbiór wszystkich form fleksyjnych leksemu:**
AllForms:- (Form*)
Form:- {VALUE_STRING}
**Zbiór wszystkich synonimów leksemu:**
Synonyms:- (Synonyme*)
Synonyme:- (Lexeme, Similarity)
Similarity:- {VALUE_DOUBLE}
**Atrybuty leksemu (część mowy i oznaczenie jednostki nazwanej):**
Attribs:- (Pos, Semantic, Id, NamedEntity, Frequency, Remark)
Pos:- {VALUE_INTEGER}
Semantic:- {VALUE_STRING}
Id:- {VALUE_INTEGER}
NamedEntity:- (NEName, NEIdent, NEIndex)
NEName:- {VALUE_STRING}
NEIdent:- {VALUE_INTEGER}
NEIndex:- {VALUE_INTEGER}
Frequency:- {VALUE_DOUBLE}
Remark:- {VALUE_STRING}
NormalizedForm:- {VALUE_STRING}
Rank:- {VALUE_DOUBLE}
PhraseText:- {VALUE_STRING}

DODATEK B

# Gramatyka szablonów pytań

## B.1. Specyfikacja formalna języka szablonów pytań

W ramach zapisu szablonów można użyć mechanizmu definiowania **makr**. Zdefiniowane makro można użyć w dowolnym miejscu pliku (po jego definicji) wpisując nazwę makra w nawiasach klamrowych. Nazwa taka zostaje zastąpiona tekstem zdefiniowanym w momencie definicji makra.

```
::TemplateRuleFile::
    (macro)*
    (template)+

::macro::
    macro_name '=' text

::macro_name::
    [A-Z_]+

::template::
    'Question:'     text
    'Match:'       (match)
    'Type:'        (type)
    'Topic:'       (topic_or_action)
    'Action:'      (topic_or_action)
    'Constraints:' (contraints)
    'Phrase:'      (phrase)

::match::
    (match_entity) (' ' (match_entity))*

::match_entity::
    '<' (match_entity_body) '>' (quantity_modifier)?

::match_entity_body::
    ('OR:')? (match_atomic_entity) (';' (match_atomic_entity))*

::quantity_modifier::
    '*' | '+' | '?' | '{' [0-9]+ ',' [0-9]+ '}'

::match_atomic_entity::
    ( (lex_property) (operation) (text) | 'TRUE' )

::lex_property::
```

```
    'used' | 'base' | 'form' | 'synonyme' | 'pos' |
    'sem'  | 'ne'   | 'normal'

::operation::
    '=' | '!=' | '~' | '!~'

::type::
    (question_type) ';' (answer_type)

::question_type::
    'TIME' | 'PLACE' | 'UNKNOWN'

::answer_type::
    'PAST' | 'PRESENT' | 'FUTURE' | 'UNKNOWN'

::topic_or_action::
    (match_reference) | 'lexeme(' text ')'

::match_reference::
    '\'[0-9]+

::constraints::
    constraint( ';' (constraint))*

::constraint::
    ( '+' | '-' ) ( (match_reference) |
         'lexeme(' text ')')

::phrase::
    (phrase_token) (';' (phrase_token))*

::phrase_token::
    text ('|' (text | '(empty)') )* |
        ((phrase_lex_property)
            '(' (text | (match_reference) | (qquery_reference) ) ')')

::phrase_lex_property::
    'used' | 'base' | 'normal' | 'form' | 'synonyme'

::qquery_reference::
    '\topic' | '\action'
```

## B.2. Przykładowe szablony pytań systemu Hipisek.pl

### B.2.1. Makra prezentowanych szablonów

```
ULET = [A-ZĄŻŚŹĘĆÓŁŃ]
LET  = [a-zA-ZążśźęćółńĄŻŚŹĘĆÓŁŃ]
ULETWORD = {ULET}{LET}+
```

```
PODAJ_OP = <normal~(?:podaj|wymień|napisz|powiedz|wyświetl)>?
```

### B.2.2. Szablon pytania o miejsce w którym znajduje się osoba x

```
Question: Gdzie-simple-person
Match:                \
       {PODAJ_OP}     \
       <normal=gdzie> \
       <normal=jest>  \
       <normal~(?:pan|pani|poseł|posłanka|sędzia|prezydent|premier|prezes)>? \
       <OR:sem=name;ne=person>+

Type: PLACE;PRESENT
Topic:  \5
Action: \3
Constraints: -\4
Phrase: base(\topic);jest|znajduje się|przebywa;w
Phrase: base(\topic); \
poleciał|poleciała|pojechał|pojechała|odjechał \
|odjechała|wyruszył|wyruszyła|wypłynął|wypłynęła|wyleciał|wyleciała;do|na
```

### B.2.3. Szablon pytania o czas rozpoczęcia wydarzenia

```
Question: Kiedy-simple-event 1
Match: {PODAJ_OP}     \
       <normal=kiedy> \
       <normal~(?:odbędzie|rozpocznie|zacznie|będzie|zaczyna|startuje)> \
       <sem~event>    \
       <normal~w>?    \
       <used~{ULETWORD}>?

Type:   TIME;FUTURE
Topic:  \4;\5;\6
Action: \3
Phrase: \topic;odbędzie|rozpocznie|zacznie|będzie|zaczyna|startuje;się
Phrase: \topic;będzie|startuje
Phrase: \topic;synonym(\action)
```

**DODATEK C**

# Gramatyka reguł systemu NER projektu Hipisek.pl

## C.1. Specyfikacja formalna języka reguł NER

W ramach zapisu reguł można użyć mechanizmu definiowania **makr**. Zdefiniowane makro można użyć w dowolnym miejscu pliku (po jego definicji) wpisując nazwę makra w nawiasach klamrowych. Nazwa taka zostaje zastąpiona tekstem zdefiniowanym w momencie definicji makra.

```
::NERRulesFile::
    (macro)*
    (rule)+

::macro::
    macro_name '=' text

::macro_name::
    [A-Z_]+

::rule::
    'Rule:'   text
    'Match:'  match
    'Action:' action

::match::
    '<' (match_entity) '>'

::match_entity::
    (match_atomic_entity) ( ';' (match_atomic_entity))*

::match_atomic_entity::
    (lex_property) (operation) text

::lex_property::
    'used' | 'base' | 'form' | 'synonyme' | 'pos' |
    'sem'  | 'normal'

::operation::
    '=' | '!=' | '~' | '!~'

::action::
    'Type=' text
```

## C.2. Przykładowe reguły NER wykorzystywane w projekcie Hipisek.pl

### C.2.1. Makra użyte w przedstawionych regułach

```
ULET = [A-ZĄŻŚŹĘĆÓŁŃ]
LLET = [a-zążśźęćółń]
LET  = [a-zA-ZążśźęćółńĄŻŚŹĘĆÓŁŃ]
ULETWORD = {ULET}{LET}+
NUM = [0-9]+
MONTH_NAME = (?:styczeń|luty|marzec|kwiecień|maj|czerwiec  \
  |lipiec|sierpień|wrzesień|październik|listopad|grudzień  \
  |stycznia|lutego|marca|kwietnia|maja|czerwca|lipca       \
  |sierpnia|września|października|listopada|grudnia)
```

### C.2.2. Reguły oznaczające osoby

```
# np. pan Marcin Walas, pani Aldona Nowak, ksiądz Jan Bosko
Rule: pan, pani 1
Match: <normal~(?:pan|pani|panna|ksiądz|prezydent|poseł|posłanka|sędzia)> \
       <used~{ULETWORD}> <used~{ULETWORD}>
Action: type=person

# np. Marcin Walas, Tomasz Nowak
Rule: person 1
Match: <sem=name> <used~{ULETWORD};sem!=city;sem!=country>
Action: type=person

# np. Walas Marcin, Nowak Tomasz
Rule: person 2
Match: <used~{ULETWORD};sem!=city;sem!=country> <sem=name>
Action: type=person
```

### C.2.3. Reguły oznaczające daty

```
# np. 12 stycznia 2009 roku
Rule: date 1
Match: <used~{NUM}> <normal~{MONTH_NAME}> <used~[0-9]{4}> <used~roku|r\.?|rok>
Action: type=date

# np. 12 stycznia
Rule: date 2
Match: <used~{NUM}> <normal~{MONTH_NAME}>
Action: type=date

# np. 12 stycznia 2009 r.
Rule: date 3
Match: <used~[0-9]{1,2}\.[0-9]{2}\.[1-9][0-9]{3}> <used~r\.?>
Action: type=date
```

**DODATEK D**

# Pytania zestawu testowego serwisu Hipisek.pl

1. Podaj gdzie pojechał Lech Wałęsa
2. Gdzie aborcja jest legalna
3. Gdzie pojechał Lech Kaczyński
4. Gdzie wykryto w Polsce wirusa grypy
5. Które miasta zorganizują mecze mistrzostw Europy
6. Gdzie zagra Doda?
7. Kiedy Polska wejdzie do strefy euro
8. Podaj miejsce spotkania Radosława Sikorskiego z prezydentem Kaczyńskim
9. Od kiedy mamy kryzys finansowy
10. Od kiedy Nergal jest z Dodą
11. Kiedy talibowie zabili polskiego inżyniera
12. Ile trwa kadencja prezydenta
13. Ile lat trwa wojna w Darfurze
14. Gdzie nie ratyfikowano traktatu lizbońskiego
15. Podaj plan podróży premiera Putina do Polski
16. Gdzie Madonna planuje ślub
17. Gdzie na Ukrainie odbędą się mecze mistrzostw Europy
18. Od kiedy matematyka będzie obowiązkowa na maturze
19. W którym kraju Unii jest najwyższe bezrobocie
20. Kiedy będą wybory do parlamentu europejskiego
21. W którym mieście miała miejsce konferencja klimatyczna
22. Jak długo Donald Tusk był w Peru
23. W którym mieście są najtańsze mieszkania
24. Który kraj jest najbardziej katolicki na świecie
25. Gdzie grasuje puma

# Bibliografia


[1] Dawid Weiss. Zdjąć na chwilę gogle. *Magazyn CHIP*, (9), 2002.

[2] Witold Abramowicz. *Filtrowanie Informacji*. Wydawnictwo Akademii Ekonomicznej w Poznaniu, 2008.

[3] Jimmy Lin and Boris Katz. Question answering from the web using knowledge annotation and knowledge mining techniques. 2003.

[4] Junlan Feng. Question answering with question answer pairs on the web. 2008.

[5] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[6] Sunita Sarawagi. Information extraction. *FnT Databases*, 1(3), 2008.

[7] Zygmunt Vetulani. Automatyczna interpretacja pytań i udzielanie odpowiedzi jako technologia multimedialna. 2002.

[8] Mark T. Maybury. Toward a question answering roadmap. 2002.

[9] *Strona podręcznika dokumentacji manual systemu GNU dla programu grep*. http://www.gnu.org/software/grep/doc/.

[10] Silvia Quarteroni and Suresh Manandhar. A chatbot-based interactive question answering system. 2007. Decalog 2007: Proceedings of the 11th Workshop on the Semantics and Pragmatics of Dialogue, pages 83–90.

[11] J. Renz and B. Nebel. *Qualitative Spatial Reasoning using Constraint Calculi*. Springer, 2007.

[12] Ulf Hermjakob. Parsing and question classification for question answering. Proceedings of the Workshop on Open-Domain Question Answering at ACL-2001, 2001.

[13] Jeffrey Pomerantz. A linguistic analysis of question taxonomies. *Journal of the American Society for Information Science and Technology*, 56(7):715–728, 2005.

[14] Eriks Sneiders. *Automated Question Answering Using Question Templates that Cover the Conceptual Model of the Database*. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2002.

[15] Boris Katz, Gary Borchardt, and Sue Felshin. Natural language annotations for question answering. In *Proceedings of the 19th International FLAIRS Conference (FLAIRS 2006)*, May 2006.

[16] Sophie Rosset, Olivier Galibert, Gabriel Illouz, and Aur'elien Max. Integrating spoken dialog and question answering: the ritel project. 2006.

[17] James Pustejovsky, Luc Belanger, Jos'e Casta, Rob Gaizauskas, Patrick Hanks, Bob Ingria, Graham Katz, Dragomir Radev, Anna Rumshisky, Antonio Sanlippo, Roser Saur, Andrea Setzer, Beth Sundheim, and Marc Verhagen. Nrrc summerworkshop on temporal and event recognition for question answering systems. Technical report, Brandeis University and ARDA, 2002.

[18] Vineet Sinha Karun Bakshi David Huynh Boris Katz David R. Karger Jimmy Lin, Dennis Quan. What makes a good answer? the role of context in question answering. Proceedings of the Ninth IFIP TC13 International Conference on Human-Computer Interaction (INTERACT 2003), 2003.

[19] Eduard Hovy, Laurie Gerber, Ulf Hermjakob, Michael Junk, and Chin-Yew Lin. Question answering in webclopedia. NIST Special Publication 500-249: The Ninth Text REtrieval Conference (TREC 9), 2000.

[20] Zhiping Zheng. Answerbus question answering system. School of Information University of Michigan, 2002.

[21] Krzysztof Jassem, Filip Graliński, and Michał Marcińczuk. Semi-supervised learning rule acquisition for named entity recognition and translation. 2008.

[22] Aleksander Buczyński and Adam Przepiórkowski. ♠ demo: An open source tool for partial parsing and morphosyntactic disambiguation. In *Proceedings of LREC*, 2008.

[23] Adam Przepiórkowski. *Powierzchniowe przetwarzanie języka polskiego*. Akademicka Oficyna Wydawnicza EXIT, 2008.

[24] Jim Cowie, Evgeny Ludovik, Hugo Molina-Salgado, Sergei Nirenburg, and Svetlana Scheremetyeva. Automatic question answering. Proceedings of the RIAO Conference, 2000.

[25] Gideon S. Mann. Fine-grained proper noun ontologies for question answering.

[26] Zygmunt Vetulani. *Corpus of consultative dialogues*. Wydawnictwo Naukowe Uniwersytetu im. Adama Mickiewicz w Poznaniu, 1990.

[27] Kevin Hemenway and Tara Calishain. *Spidering Hacks*. O'Reilly, 2003.

[28] Eric Brill, Susan Dumais, and Michele Banko. An analysis of the askmsr question-answering system. In *Proceedings of EMNLP 2002*, 2002.

[29] *Sphinx reference manual*. http://www.sphinxsearch.com.

[30] Filip Graliński. Wstępujący parser języka polskiego na potrzeby systemu POLENG. *Speech and Language Technology. Technologia Mowy i Języka*, t. 6, cz. III:263–276, 2002.