Uniwersytet im. Adama Mickiewicza
Wydział Matematyki i Informatyki

Tomasz Kowalski

Pozyskiwanie reguł tłumaczenia z korpusów oznaczonych składniowo

Rozprawa doktorska

Promotor: dr hab. Krzysztof Jassem

POZNAŃ 2011

Spis treści

Rozdział 1

Wprowadzenie

Tłumaczenie tekstów pomiędzy *językami naturalnymi*[1] jest zadaniem, które od ponad pół wieku[2] próbuje się realizować z wykorzystaniem komputerów, a które, wydawać by się mogło, wymaga „myślenia i inteligencji". Jednak dopiero w ostatnim dziesięcioleciu prace nad automatycznym wytwarzaniem przekładów tekstów z jednego języka naturalnego na drugi bez udziału człowieka, czyli nad tak zwanym *tłumaczeniem automatycznym*[3], zaczęły odznaczać się szczególnym dynamizmem. Narastające przeświadczenie o tym, że język naturalny jest tak bogaty, że nigdy nie uda się go w pełni zanalizować oraz praktyczne trudności związane z zakodowaniem opisu języka w postaci programu komputerowego, ukierunkowały badania naukowe na umożliwienie maszynie tego, aby sama nauczyła się tłumaczyć. Dąży się do tego, aby tłumaczenie automatyczne dokonywało się na podstawie analizy dużych zbiorów przetłumaczonych wcześniej tekstów, a więc tak zwanych *tekstów równoległych*.

Celem tej pracy jest przedstawienie (w rozdziale 3) i weryfikacja (w rozdziale 4) metody analizy zbioru tekstów, złożonego z oryginału oraz jego tłumaczenia na język polski. Wynikiem owej analizy jest zbiór *reguł* przeznaczonych do zastosowania przy tłumaczeniu automatycznym tekstów na język polski. Oprócz algorytmu analizy zbioru tekstów, w tej pracy przedstawiony został również algorytm, który stosuje pozyskane reguły dla otrzymania przekładu tekstu na język polski.

W rozdziale 2 zostały przedstawione wybrane algorytmy tłumaczenia automatycznego. Ich prezentacja rozpoczyna się od wyróżnienia trzech paradygmatów obecnych w tej dziedzinie (rozdział 2.1).

---

1. Przez *język naturalny*, należy rozumieć każdy język używany przez ludzi na co dzień — w mowie lub piśmie.

2. Na uniwersytecie w Georgetown (USA) zaprezentowano w 1954 roku system tłumaczący 49 wyselekcjonowanych zdań rosyjskich na język angielski.

3. *Tłumaczenie automatyczne* zamiennie zwane jest również *tłumaczeniem maszynowym*.

Są to algorytmy:

- motywowane lingwistycznie,
- niemotywowane lingwistycznie,
- hybrydowe.

Spośród znanych algorytmów motywowanych lingwistycznie, w rozdziale 2.2 opisane zostały algorytmy stosowane w systemie tłumaczenia automatycznego POLENG. System ten potrafi tłumaczyć teksty na język polski, a część stosowanych w nim algorytmów, wykorzystujących tzw. atrybutywne gramatyki bezkontekstowe [4], została użyta w rozwiązaniach autorskich prezentowanych w rozdziale 3.

Spośród znanych algorytmów niemotywowanych lingwistycznie, w rozdziale 2.3 opisane zostały pewne algorytmy stosujące modelowanie statystyczne dla automatycznego generowania przekładów tekstów. Algorytmy te są niezależne od języka naturalnego, do którego są stosowane i również zostały użyte w rozwiązaniach autorskich prezentowanych w rozdziale 3.

Spośród znanych algorytmów z grupy hybrydowych, a to właśnie do niej należy zaliczyć algorytmy opisane w rozdziale 3, przedstawione zostały dwa rozwiązania.
W pierwszym z nich (rozdział 2.4.1), na bazie zbioru tekstów przetłumaczonych z języka chińskiego na język angielski, dla wykazania szkodliwości ograniczeń przyjmowanych przy implementacji algorytmów tłumaczenia automatycznego wykorzystujących czyste modelowanie statystyczne, użyto podobnej struktury danych, jak w rozwiązaniach prezentowanych w rozdziale 3. Struktura ta składa się ze *zdania źródłowego* [5] i *zdania docelowego* [6] poddanego analizie składniowej.
Drugie omawiane w tej grupie rozwiązanie (rozdział 2.4.2) dotyczy algorytmów tłumaczenia na język polski, w których również użyto podobnej struktury danych, jak w rozwiązaniach prezentowanych w rozdziale 3, jednak z prowadzeniem analizy składniowej zdania źródłowego, a nie docelowego.

W rozdziale 3 przedstawiono szczegółowy opis algorytmów umożliwiających „nauczenie się" tłumaczenia tekstów na język polski, przy ograniczonej ilość dostępnych „danych treningowych". Rozdział 3.4 poświęcony został omówieniu algorytmu uczącego, a więc pozyskującego reguły przeznaczone do tłumaczenia automatycznego na język polski, z niewielkich korpusów dwujęzycznych oznaczonych składniowo. Rozdział 3.5 skupia się na prezentacji algorytmu tłumaczącego, a więc stosującego reguły pozyskane przez algorytm uczący, do tłumaczenia automatycznego na język polski.

4. Atrybutywna gramatyka bezkontekstowa jest gramatyką bezkontekstową rozszerzoną o parametry i operacje na nich

5. *Zdanie źródłowe* to zdanie w *języku źródłowym*, a więc w języku, z którego dokonuje się tłumaczenia.

6. *Zdanie docelowe* to zdanie będące tłumaczeniem zdania źródłowego na *język docelowy*.

W rozdziale 4 przeprowadzono weryfikację poprawności algorytmów prezentowanych w rozdziale 3. Omówiono praktyczne zagadnienia związane z ich implementacją (rozdział 4.2) oraz wyniki eksperymentów mających na celu określenie pewnych własności tejże implementacji (rozdział 4.3).

Rozdział 2

Algorytmy tłumaczenia automatycznego

Dynamikę badań naukowych najlepiej ilustruje ilość publikacji w danej dziedzinie. W przypadku tłumaczenia automatycznego dobrze widać to na przykładzie tzw. *tłumaczenia statystycznego* — jak dotąd opublikowano na ten temat około tysiąca prac, z czego mniej więcej połowę w ostatnich trzech latach [17].

W niniejszym rozdziale przedstawiono tylko te algorytmy tłumaczenia automatycznego, które mają ścisły związek z algorytmami prezentowanymi w rozdziale 3. W pierwszej kolejności przedstawiona została pewna systematyka algorytmów tłumaczenia, a następnie omówione zostały algorytmy reprezentatywne dla poszczególnych klas.

2.1. Typy algorytmów tłumaczenia automatycznego

Już od dziesięciu lat wśród algorytmów tłumaczenia automatycznego dostrzec można trzy paradygmaty [7]:

- systemy stosujące wiedzę lingwistyczną,
- systemy nie stosujące wiedzy lingwistycznej,
- systemy hybrydowe.

Systemami motywowanymi lingwistycznie określono grupę algorytmów, które zostały zaprojektowane tak, aby w oparciu o dobrze znane ograniczenia składniowe, leksykalne i semantyczne, wygenerować wyrażenie w języku docelowym, odpowiadające zdaniu źródłowemu.

Cechami charakterystycznymi algorytmów z tej grupy są:

- wykorzystywanie bogatego słownika,
- 3–etapowy proces tłumaczenia.

Informacje wykorzystywane przez algorytmy z tej klasy zawarte są w:

- słowniku mogącym zawierać szereg informacji morfologicznych, syntaktycznych i semantycznych na temat elementów języka źródłowego i docelowego, oraz ich wzajemnych relacji,
- bazie reguł sterujących procesem tłumaczenia.

Proces tłumaczenia realizowany przez algorytmy z tej grupy składa się z następujących etapów:

1. analizy tekstu w języku źródłowym, zwanej także *parsowaniem*,
2. konwersji, zwanej także *transferem*, wiedzy o tekście źródłowym (dostępnej w postaci pewnych struktur zależnych od języka źródłowego), na wiedzę o tekście, który należy wytworzyć w języku docelowym,
3. syntezy tekstu w języku docelowym, zwanej także *generowaniem*.

Szczegółowej klasyfikacji algorytmów motywowanych lingwistycznie dokonuje się w zależności od nacisku położonego na realizację każdego w powyższych etapów oraz typu informacji lingwistycznych zawartych w słowniku i bazie reguł.

Systemami niemotywowanymi lingwistycznie określono grupę algorytmów, które nie wykorzystują jawnie teorii lingwistycznych, ani nawet lingwistycznych właściwości konkretnego języka, lecz są zdolne „nauczyć się tłumaczenia" na podstawie prezentowanych przykładów.

Cechą charakterystyczną algorytmów z tej grupy jest założenie dostępności dużych korpusów tekstów równoległych, które mogą zostać użyte jako zbiory treningowe lub wprost jako baza wiedzy. Wyróżnia się dwie podgrupy algorytmów niemotywowanych lingwistycznie:

- algorytmy tłumaczenia statystycznego — wykorzystujące korpusy równoległe do trenowania parametrów modeli statystycznych,
- algorytmy tłumaczenia przez analogię — traktujące korpusy równoległe jako bazy przykładów — wzorców tłumaczenia do zastosowania w podobnych okolicznościach.

Systemami hybrydowymi określono grupę algorytmów, w których wykorzystuje się pewne metody przetwarzania opracowane na potrzeby algorytmów motywowanych lingwistycznie zestawiając je z pewnymi metodami przetwarzania charakterystycznymi dla metod niemotywowanych lingwistycznie.

Najpowszechniejszą konfiguracją hybrydową jest modyfikacja procesu tłumaczenia stosowanego w metodach motywowanych lingwistycznie. Istotna zmiana polega w tym przypadku na realizacji transferu pomiędzy strukturami pośrednimi nie za pomocą algorytmów regułowych lecz przy użyciu algorytmów tłumaczenia statystycznego lub przez analogię.

Algorytm autorski opisany w rozdziale 3 należy zaliczyć również do algorytmów hybrydowych. Stosowany jest w nim 3-etapowy proces tłumaczenia (taki jak w tłumaczeniu motywowanym lingwistycznie), lecz reguły transferu oraz słownik systemu przygotowywane są w sposób charakterystyczny dla algorytmów tłumaczenia statystycznego: wykorzystując korpusy równoległe do trenowania modeli statystycznych.

2.2. Klasyczne algorytmy tłumaczenia motywowanego lingwistycznie

Systemy tłumaczenia automatycznego motywowane lingwistycznie zbudowane są z algorytmów zorientowanych przede wszystkim na dostarczenie poprawnego składniowo tekstu w języku docelowym, otrzymanego poprzez konwersję struktury składniowej tekstu w języku źródłowym. [7]

Przykładem systemu tego typu może być system Poleng. Opis przetwarzania tekstów polskich w tym systemie opublikowany został w 2006 roku [12].

2.2.1. Reprezentacja danych leksykalnych

System tłumaczący korzysta z wiedzy lingwistycznej zawartej w *bazie leksykalnej* (zwanej również słownikiem), gdzie przechowywane są informacje morfologiczne, syntaktyczne i semantyczne na temat elementów języka źródłowego i docelowego, oraz ich wzajemnych relacji.

Struktura bazy leksykalnej

Podstawowym problemem w skonstruowaniu bazy leksykalnej jest zdefiniowanie elementów języka opisywanych w bazie. W systemie Poleng wyróżniono następujące elementy języka: wyraz, wielowyrazowa jednostka leksykalna oraz fraza leksykalna.

Definicja 1. (Wyraz) *Element języka naturalnego (np. języka polskiego) pisany bez spacji nazywamy (pojedynczym)* wyrazem.

Przykład. Wyrazami są: „prawo", „pobiłem". Nie jest wyrazem: ciąg „na pewno" (są to dwa wyrazy), ciąg „brkjs" (ciąg ten nie należy do żadnego języka naturalnego).

Definicja 2. (Wielowyrazowa jednostka leksykalna) *Ciąg co najmniej dwóch, następujących bezpośrednio po sobie wyrazów, które w procesie przetwarzania komputerowego należy traktować jako niepodzielną jednostkę języka, nazywamy* wielowyrazową jednostką leksykalną.

Przykład. Przykładowymi wielowyrazowymi jednostkami leksykalnymi są: „na przykład", „na pewno", „dlatego że".

Definicja 3. (Fraza leksykalna) *Grupę co najmniej dwóch wyrazów, które nie tworzą wielowyrazowej jednostki leksykalnej, nazywamy* frazą leksykalną *gdy:*

- *grupa ma taką samą funkcję składniową jak pojedynczy wyraz,*
- *znaczenie grupy (określane na przykład w* słowniku dwujęzycznym *ekwiwalentem w innym języku), nie jest prostym złożeniem znaczeń jej elementów składowych.*

Przykład. Frazą leksykalną jest fraza „łamać prawo". Pełni ona funkcję składniową taką jak pojedynczy czasownik. Frazą leksykalną jest też raza „rekord świata". Jej funkcja składniowa jest taka sama jak pojedynczego rzeczownika.

Frazy takie, w odróżnieniu od wielowyrazowych jednostek leksykalnych, mogą być przedzielone w zdaniu innymi wyrazami (np. „łamać często prawo") lub tworzyć formy fleksyjne poprzez zamianę końcówki wyrazów wewnętrznych (np. „rekordu świata").

Definicja 4. (Jednostka słownikowa) *Wyraz, wielowyrazową jednostka leksykalną lub frazę leksykalną, nazywamy* jednostką słownikową.

W bazie leksykalnej (słowniku) systemu tłumaczenia każdej jednostce słownikowej przydzielony jest jej opis, przeznaczony do wykorzystania przez algorytm tłumaczący.

Definicja 5. (Leksem) *Zbiór wyrazów, wielowyrazowych jednostek leksykalnych lub fraz leksykalnych, podlegających wspólnemu opisowi nazywamy* leksemem.

Jednostki słownikowe wchodzące w skład leksemu nazywamy formami fleksyjnymi leksemu.

Przykład. Leksemem jest:

- zbiór jednoelementowy {przy}; jest to jedyna formy fleksyjnej leksemu przyimka „przy",
- zbiór dwuelementowy {przed, przede}; są to wszystkie formy fleksyjne leksemu przyimka „przed",
- zbiór {komputer, komputera, komputerowi, ...}; zbiór ten zawiera wszystkie formy odmienione leksemu rzeczownika „komputer".
- zbiór {łamałem prawo, łamię prawo, ...}; zbiór ten zawiera wszystkie formy odmienione leksemu frazy „łamać prawo".

Definicja 6. (Słownik) *Listę struktur (haseł) złożonych z:*

- *leksemu (reprezentowanym albo jawnie przez wszystkie formy fleksyjne, albo tylko przez formę podstawową wraz z algorytmym generowania z niej form fleksyjnych),*
- *zbioru informacji przeznaczonych do wykorzystania przez algorytm tłumaczący,*

nazywamy słownikiem systemu tłumaczenia.

Przykład. W słowniku systemu POLENG z każdym leksemem jednego z języków podane są informacje o odpowiednikach w drugim języku, wzorce lub dopełnienia składniowe związane z leksemem oraz powiązania semantyczne z pewną ustaloną hierarchią bytów opisującą dziedzinę tłumaczenia.

Algorytm tworzenia bazy leksykalnej

W klasycznych systemach tłumaczenia motywowanego lingwistycznie słowniki systemu tłumaczącego przygotowywane są przez ludzi. Zadanie to wymaga wypracowania kompromisu pomiędzy wielkością słownika i dokładnością opisu jego haseł z jednej strony, a dostępną ilością zasobów ludzkich z drugiej strony. W systemie POLENG kompromis ten osiągnięto poprzez przyjęcie następujących założeń:

- jednostki słownikowe, których częstość występowania w korpusach tekstowych jest wysoka, zostały opisane przez leksykografów zgodnie z ich najlepszą wiedzą,
- jednostki słownikowe, których częstość występowania jest niska, zostały wygenerowane automatycznie poprzez automatyczną konwersję danych ze słowników tradycyjnych Wydawnictwa Naukowego PWN (np. WIELKI SŁOWNIK ANGIELSKO–POLSKI [20]) do formatu słownika systemu POLENG.

Przygotowanie słownika dla systemu POLENG odbyło się zatem w następujących etapach:

1. przygotowanie i selekcja korpusów tekstowych,
2. *lematyzacja korpusu*,
3. stworzenie listy frekwencyjnej wyrazów i ustalenie częstościowego progu przeznaczenia hasła do opisu ręcznego,
4. ręczne opisanie haseł „częstych” na podstawie dostępnych słowników tradycyjnych i analizy ich wystąpień w korpusach tekstów,
5. automatyczne wygenerowanie opisów haseł „rzadkich”.

Przygotowanie korpusów wiąże się najczęściej z przetworzeniem tekstów pochodzących z różnych źródeł do jednolitego formatu, który umożliwi ich dalsze przetwarzanie.

Teksty wchodzące w skład korpusu mogą pochodzić z różnych źródeł. Zwykle oznacza to, że zapisane są w postaci elektronicznej z użyciem różnych formatów danych np. stron internetowych (`html`), dokumentów biurowych edytorów tekstu (`odt`, `doc`, `rtf`), dokumentów gotowych do wydruku (`pdf`, `ps`), itd.

Dla dalszego przetwarzania tych tekstów należy sprowadzić je do wspólnego formatu, co może wiązać się z koniecznością np. ujednolicenia kodowania znaków, eliminacją elementów graficznych, eliminacją graficznego formatowania tekstu, itp.

Lematyzacji korpusu dokonuje się za pomocą *analizatora morfologicznego*, który jest wyspecjalizowanym narzędziem zdolnym do wyodrębnienia wyrazów w tekście i oznaczenia leksemów, do których należą. Dzięki lematyzacji możliwe staje się ustalenie częstości występowania leksemów w korpusie.

Listy frekwencyjne leksemów sporządzane są w celu ograniczenia liczby haseł słownika opisywanych ręcznie oraz w celu rozstrzygania niejednoznaczności leksykalnych.

W przypadku wyrazów, które można traktować jako formę fleksyjną więcej niż jednego leksemu, analiza częstości wystąpień pozwala wybrać to znaczenie, które jest istotnie częściej spotykane niż pozostałe.

Ręczne opisanie haseł wiąże się przede wszystkim z umieszczeniem w słowniku przeznaczonym do tłumaczenia automatycznego szczegółowych informacji o charakterystyce leksemu w języku źródłowym oraz o charakterystyce jego odpowiednika w języku docelowym.

2.2.2. Algorytmy analizy składniowej

Pierwszym etapem działania algorytmów tłumaczenia umotywowanych lingwistycznie jest analiza składniowa tekstu źródłowego.

Definicja 7. (Analiza składniowa) *Proces mający na celu określenie reprezentacji zdania, w której poszczególnym segmentom zdania przypisana jest ich funkcja, jaką pełnią w zdaniu (nazywana funkcją składniową), nazywać będziemy* analizą składniową *lub* parsowaniem.

Reprezentacją zdania źródłowego może być na przykład tzw. *drzewo zależności* [23], tzw. *struktura cech* [3] lub drzewo struktury składniowej — stosowane m.in. w systemie Poleng.

Definicja 8. (Drzewo struktury składniowej) *Reprezentację wyrażenia w języku naturalnym mającą postać drzewa, w którym każdy węzeł odpowiada pewnemu (najczęściej spójnemu) fragmentowi zdania, pełniącemu określoną funkcję składniową, nazywamy* drzewem struktury składniowej *lub* drzewem parsowania.

Definicja 9. (Zstępująca strategia parsowania) *Analiza składniowa, w której a priori zakłada się, że rozpatrywane zdanie posiada określoną strukturę, nazywamy* zstępującą strategią parsowania.

Przykład. Zstępująca strategia parsowania może zostać zrealizowana np. przy użyciu reguł gramatyki, które zostały zapisane zgodnie z formalizmem DCG [24] w postaci programu prologowego. Analiza składniowa może być w takim przypadku realizowana przez algorytm wnioskowania stosowany przez interpreter języka Prolog [1].

Program prologowy rozpoznający angielskie zdanie „a cat eats a mouse" może mieć postać:

```
sentence(S1,S3) :- noun_phrase(S1,S2), verb_phrase(S2,S3).
noun_phrase(S1,S3) :- det(S1,S2), noun(S2,S3).
verb_phrase(S1,S3) :- verb(S1,S2), noun_phrase(S2,S3).
det([a|X], X).
noun([cat|X], X).
noun([mouse|X], X).
verb([eats|X], X).
```

Definicja 10. (Wstępująca strategia parsowania) *Analiza składniowa, która polega na iteracyjnym określaniu funkcji składniowych dla coraz większych (w sensie ilości wyrazów) fragmentów zdania, nazywamy* wstępującą strategią parsowania.

Przykład. Wstępująca strategia parsowania może zostać zrealizowana np. przy użyciu reguł zgodnych z formalizmem pozwalającym opisać tzw. *atrybutywną gramatykę bezkontekstową*, a więc gramatykę bezkontekstową rozszerzoną o parametry i operacje na nich [6]. Analiza składniowa może być w tym przypadku realizowana przy użyciu algorytmu CYK [11].

W początkowych wersjach systemu „Poleng" stosowano zstępującą strategię parsowania, realizowaną za pomocą mechanizmu języka Prolog. Strategia ta okazała się nieefektywna i zastąpiono ją strategią wstępującą [10].

Przykładowa reguła atrybutywnej gramatyki bezkontekstowej dla języka polskiego, rozpoznająca konstrukcję rzeczownikową z przydawką dopełniaczową (oznaczoną przez `krzdop`) ma w systemie Poleng postać:

```
krzdop = katr* szfrz %katr[+szfrz:przyddop]% szfrz.P == dop
```

1. *Prolog* (od francuskiego Programmation en Logique) jest to beztypowy, deklaratywny język programowania logicznego, stworzony w 1971 roku. Opiera się on o rachunek predykatów pierwszego rzędu ograniczony do klauzul Horna. Dostępny m.in. jako: SWI-Prolog (`http://swi-prolog.org`), GNU Prolog (`http://gprolog.org`.

Reguła ta składa się z trzech części:
- produkcji `krzdop = katr* szfrz`, gdzie `katr` oznacza konstrukcję atrybutywną, a `szfrz` – szereg fraz rzeczownikowych,
- instrukcji budowania drzewa struktury składniowej `%katr[+szfrz:przyddop]%`, która oznacza, że konstruowane drzewo ma składać się z drzewa reprezentującego konstrukcję atrybutywną `katr`, do którego z prawej strony (`+`) dołączone zostanie drzewo reprezentujące szereg fraz rzeczownikowych `szfrz`, opatrzone etykietą przydawki dopełniaczowej `:przyddop`,
- wyrażenia opisującego operacje wykonywane na atrybutach `szfrz.P == dop`, które oznacza, że reguła ma zastosowanie wyłącznie, gdy szereg fraz rzeczownikowych `szfrz` występuje w dopełniaczu, a więc gdy atrybut `P` określający przypadek ma wartość `dop`.

Powyższa reguła opisuje takie konstrukcje jak na przykład „komputerowi osobistemu mojego brata".

Definicja 11. (Reguły leksykalne) *Najbardziej podstawowe reguły gramatyki używanej do analizy leksykalnej, które odwołują się do symboli terminalnych, czyli wyrazów języka, nazywamy* regułami leksykalnymi.

Przykład. Reguła leksykalna zapisana w postaci programu prologowego, opisująca angielski rzeczownik „cat", może mieć następującą postać:

```
noun([cat|X], X).
```

Dla pokrycia całego słownictwa języka naturalnego potrzeba od setek tysięcy do kilku milionów reguł leksykalnych. □

Wynikiem analizy składniowej dla zdania może być:
- jedno drzewo struktury składniowej obejmujące całe zdanie,
- wiele drzew reprezentujących różne interpretacje składniowe zdania,
- brak drzew obejmujących całe zdanie; wynikiem analizy jest las, w którym drzewa obejmują fragmenty zdania.

Od systemu tłumaczenia automatycznego oczekuje się jednoznacznego wyniku tłumaczenia. Implikuje to konieczność wyznaczenia jednej, „najlepszej", reprezentacji zdania, które podlega dalszemu przetwarzaniu przez system tłumaczenia.

Przykład. W systemie POLENG najlepsza reprezentacja zdania ustalana jest w oparciu o dwa kryteria:
- najdłuższego początku,
- najdłuższej frazy słownikowej.

Kryterium najdłuższego początku stosowane jest dla zdania lub jego fragmentu, które nie zostało sparsowane w całości. Wybierana jest taka reprezentacja, w którym sparsowany został najdłuższy podciąg z lewej strony. Dla pozostałej części zdania stosuje się rekurencyjnie tą samą procedurę.

Kryterium najdłuższej frazy słownikowej ma zastosowanie dla określenia najlepszej spośród reprezentacji obejmujących ten sam fragment tekstu. Wybierana jest wtedy reprezentacja zawierająca poddrzewo, odpowiadające frazie ze słownika o największej liczbie wyrazów.

2.2.3. Algorytm transferu automatycznego

Drugim etapem przetwarzania w systemie tłumaczenia motywowanego lingwistycznie jest transfer najlepszej reprezentacji składniowej języka składniowego na reprezentację składniową drzewa docelowego. Transfer ten dokonuje się w oparciu o reguły zapisane w pewnym formalizmie, przetwarzalnym przez komputer. Transfer zostanie przedstawiony na przykładzie systemu Poleng.

Język opisu reguł zastosowany w systemie Poleng można zdefiniować za pomocą następującej gramatyki bezkontekstowej.

Symbole terminalne:

- wyrazy funkcyjne: `if`, `and`, `&&`, `or`, `||`, `not`, `!`, `else`, `sub`, `foreach`, `return`, `among`,
- referencje do węzłów drzewa języka źródłowego; są to napisy zaczynające się od znaku `@`,
- referencje do węzłów drzewa języka docelowego; są to napisy zaczynające się od znaków `%` lub `%%`,
- zmienne, których wartościami nazwy węzłów drzewa języka docelowego; są to napisy zaczynające się od znaku `$`,
- atomy alfanumeryczne i liczb określające wartości atrybutów węzłów drzew (np. cech gramatycznych),
- nazwy atrybutów węzłów drzew struktury składniowej,
- symbole operatorów: `!=`, `==`, `:=`, `->`, `=>`, `::`, `(`, `)`, `{`, `}`, `[`, `]`.

Reguły zapisywane są jako ciągi instrukcji. Instrukcja języka reguł może mieć postać:

- instrukcji prostej (np. instrukcji przypisania),
- instrukcji złożonej (ciągu instrukcji zamkniętych nawiasami klamrowymi),
- instrukcji warunkowej (o składni zgodnej z językiem C),
- instrukcji przypisania referencji,
- instrukcji zwrócenia wyniku przez podprogram.

Reguły transferu odpowiadają za przekształcenie drzewa składniowego wyrażenia w języku źródłowym w odpowiadające mu drzewo składniowe wyrażenia w języku docelowym. Umożliwiają one następujące przekształcenia:

- zmianę nazwy węzła terminalnego i nieterminalnego,
- zmianę etykiety węzła,
- ustalenie wartości atrybutów cech gramatycznych i składniowych,
- dołączenie i usunięcie węzła i gałęzi drzewa.

Przykład. Usunięcie węzła zaimka zwrotnego „się", w przypadku gdy odpowiednikiem czasownika zwrotnego w języku źródłowym jest czasownik niezwrotny w języku docelowym, możliwe jest dzięki regule:

```
delete(,oneself,).
```

Reguła, która umożliwia dodanie informacji o zwrotności czasownika w języku docelowym, wygląda następująco:

```
if (Refl)
{
    insert_right(,'oneself',);
}
```

Przykład. Reguła umożliwiająca transfer zdań podrzędnych stanowiących realizację akomodacji czasownika (zgodnie z informacją o czasowniku pochodzącą ze słownika) może zostać zapisana następująco:

```
if ($COMPL1.Cat=SubC)
{
    $COMPL1.Type:=TargetCategory1;
    $COMPL1.Prep:=Prep1;
}
```

Reguła taka pozwala np. na poprawne przetłumaczenie zdania: „Chcę, abyś dał mi pieniądze" na zdanie „I want you to give me money".

2.2.4. Algorytmy syntezy

Proces syntezy wyrażenia w języku docelowym z drzewa struktury składniowej otrzymanego po fazie transferu zostanie przedstawiony na przykładzie systemu POLENG.

Reguły modyfikacji składniowej mają na celu dostosowanie drzewa struktury składniowej do zasad składni języka docelowego. Reguły zapisane są przy użyciu tego samego formalizmu co reguły transferu.

Przykład. Reguła modyfikacji składniowej dla języka angielskiego, która ustala pożądaną kolejność poddrzew drzewa o korzeniu w węźle `VP` może zostać zapisana następująco:

```
%PreSubjectAdv < %PreSubjectClause < %Question < %Subject <
%Disjunct < %Head < %Oneself < %Compl1 < %Compl2 < %Compl3 <
%AdvManner < %AdvLoc < %AdvTime < %SubC;
```

Reguła powyższa w zastosowaniu do zdania oznajmującego informuje, że pożądana kolejność składników zdania docelowego (tu: angielskiego) jest następująca:

> Podmiot Orzeczenie Dysjunkt Czasownik ZaimekSię Dopełnienie1 Dopełnienie2 Dopełnienie3 OkolicznikSposobu OkolicznikMiejsca OkolicznikCzasu ZdaniePodrzędne

Reguły syntezy morfologicznej mają na celu wygenerowanie form fleksyjnych wyrazów języka docelowego. Język opisu tych reguł zawiera wyrażenia regularne.

Przykład. Reguła syntezy morfologicznej, która opisuje sposób regularnego tworzenia formy ciągłej czasowników angielskich w zależności od końcówki formy podstawowej, może zostać zapisana następująco:

```
if ($lex=~/[^eiyo]e$/)
{
    $lex:=substring($lex,-1)+'ing';
}
else if ($lex=~/ie$/)
{
    $lex:=substring($lex,-2)+'ying';
}
else if ($lex=~/ic$/)
{
    $lex:=$lex+'king';
}
else
{
    $lex:=$lex+'ing';
}
```

Reguła powyższa sprawdza ostanie znaki formy podstawowej czasownika angielskiego, odpowiednio ją modyfikuje i dołącza na jej koniec morfem „ing".

2.3. Klasyczne algorytmy tłumaczenia niemotywowanego lingwistycznie

Praktyka pracy tłumacza pokazuje, że tłumaczenie tekstu z jednego języka naturalnego na drugi wiąże się z wieloma decyzjami, które wydają się być trudne do sformalizowania. Ten fakt stał się podstawą prac badawczych, które miały na celu opracowanie metod „uczenia się" tłumaczenia przez komputer na podstawie już przetłumaczonych tekstów.

W późnych latach 80. w laboratoriach IBM Research, w związku sukcesem metod statystycznych w rozpoznawaniu mowy, zrodził się pomysł statystycznego tłumaczenia tekstu.

W tym rozdziale zostaną przedstawione algorytmy tłumaczenia statystycznego pochodzące z prac wykonanych w ramach projektu IBM Candidate [17].

2.3.1. Modele oparte o wyrazy

W niniejszym rozdziale zostaną przedstawione modele tłumaczenia IBM 1–5, w kolejności od najmniej do najbardziej złożonego. Każdy kolejny model rozszerza poprzedni w sposób taki, aby „uchwycić" więcej zjawisk, jakie zachodzą w procesie tłumaczenia:

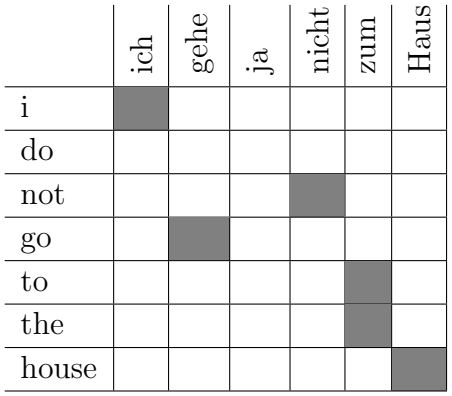
- IBM Model 1 odpowiada tłumaczeniu pojedynczych wyrazów,
- IBM Model 2 uwzględnia kolejność wyrazów w zdaniu,
- IBM Model 3 obejmuje przypadki, gdy pewne wyrazy nie tłumaczą się „jeden do jednego",
- IBM Model 4 pozwala na tłumaczenie grup wyrazów (fraz),
- IBM Model 5 jest optymalizacją IBM Model 4 zapobiegającą „marnowaniu" masy prawdopodobieństwa na niemożliwe (z perspektywy zastosowań) wyniki.

Przyjmuje się następujące oznaczenia:
$\mathbf{f} = (f_1, f_2, \ldots, f_{l_f})$ zdanie źródłowe o długości l_f
$\mathbf{e} = (e_1, e_2, \ldots, e_{l_e})$ zdanie docelowe o długości l_e

Definicja 12. (Funkcja wyrównania) *Funkcję $a : j \to i$, gdzie j jest pozycją wyrazu w zdaniu docelowym, a i pozycją wyrazu w zdaniu źródłowym, nazywamy* funkcją wyrównania.

Przykład. Dla niemieckiego zdania „ich gehe ja nicht zum Haus" i jego angielskiego tłumaczenia „i do not go to the house" funkcja wyrównania może zostać określona w następujący sposób:

$$a : \{1 \to 1, 2 \to 0, 3 \to 4, 4 \to 2, 5 \to 5, 6 \to 5, 7 \to 6\}$$



Rysunek 2.1. Graficzna reprezentacja funkcji wyrównania zdania źródłowego i docelowego z dokładnością do wyrazu.

Definicja 13. (Prawdopodobieństwo tłumaczenia leksykalnego) *Prawdopodobieństwo przetłumaczenia słowa źródłowego f jako słowa docelowego e nazywamy* prawdopodobieństwem tłumaczenia leksykalnego *i oznaczamy przez $t(e|f)$.*

Przykład. Słownik Harper Collins German Dictionary dla niemieckiego wyrazu „Haus" podaje następujące angielskie odpowiedniki: „house", „building", „home", „household", „shell".

Na podstawie analizy częstości wystąpień angielskich odpowiedników niemieckiego wyrazu źródłowego w pewnym korpusie tekstów, prawdopodobieństwo tłumaczenia leksykalnego może zostać określone następująco:

$$t(e|\text{Haus}) = \begin{cases} 0.8 & \text{dla } e = \text{house} \\ 0.16 & \text{dla } e = \text{building} \\ 0.02 & \text{dla } e = \text{home} \\ 0.015 & \text{dla } e = \text{household} \\ 0.005 & \text{dla } e = \text{shell} \end{cases}$$

Definicja 14. (IBM Model 1) *Prawdopodobieństwo, że zdanie $\boldsymbol{e}$ jest tłumaczeniem zdania $\boldsymbol{f}$, takim że wyraz e_j jest wyrównany do wyrazu wyrazu f_i zgodnie z funkcją wyrównania $a : j \to i$, określa się następująco:*

$$p(\boldsymbol{e}, a|\boldsymbol{f}) = \frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} t(e_j|f_{a(j)})$$

gdzie ϵ to stała normalizacyjna, taka by $\sum_{e,a} p(\boldsymbol{e}, a|\boldsymbol{f}) = 1$, a $(l_f+1)^{l_e}$ to liczba wszystkich możliwych wyrównań zdania źródłowego do docelowego.

Przykład. IBM Model 1 realizuje tzw. *tłumaczenie wyraz po wyrazie*. Polega ono na zastąpieniu każdego wyrazu źródłowego jego najbardziej prawdopodobnym odpowiednikiem w języku docelowym.

Definicja 15. (Prawdopodobieństwo wyrównania) *Prawdopodobieństwo, że słowo znajdujące się na pozycji i w zdaniu źródłowym $\boldsymbol{f}$ jest tłumaczone na słowo znajdujące się na pozycji j w zdaniu docelowym $\boldsymbol{e}$ nazywamy* prawdopodobieństwem wyrównania *i oznaczamy przez $a(i|j, l_e, l_f)$.*

Przykład. Następujące tłumaczenia są równie prawdopodobne według IBM Model 1, gdyż każdy wyraz źródłowy w obu przypadkach został przetłumaczony tak samo:

	natürlich	ist	das	Haus	klein
of	■				
course	■				
the			■		
house				■	
is		■			
small					■

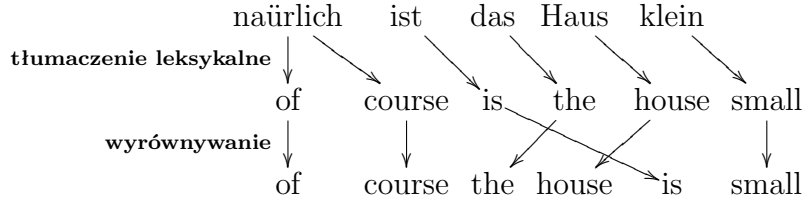
	natürlich	ist	das	Haus	klein
the			■		
course	■				
small					■
is		■			
of	■				
house				■	

Oczywiście, tłumaczenie po lewej stronie powinno być preferowane ponad tłumaczenie przedstawione po prawej stronie, ze względu na szyk zdania angielskiego. Określenie prawdopodobieństwa wyrównania służy sformalizowaniu tej preferencji.

Definicja 16. (IBM Model 2) *Prawdopodobieństwo, że zdanie $\boldsymbol{e}$ jest tłumaczeniem zdania $\boldsymbol{f}$, określa się następująco:*

$$p(\boldsymbol{e}, a|\boldsymbol{f}) = \epsilon \prod_{j=1}^{l_e} t(e_j|f_{a(j)}) a(a(j)|j, l_e, l_f)$$

Przykład. Wprowadzenie prawdopodobieństwa wyrównania powoduje, że tłumaczenie w IBM Model 2 można postrzegać jak dwufazowy proces.



W pierwszym kroku następuje tłumaczenie leksykalne, a w drugim kroku następuje zmiana kolejności wyrazów zdania. W powyższym przykładzie tłumaczenie niemieckiego czasownika „ist" na jego angielski odpowiednik dokonuje się z prawdopodobieństwem $t(\text{is}|\text{ist})$, przy prawdopodobieństwie wyrównania $a(2|5,6,5)$ — piąte słowo w sześciowyrazowym zdaniu angielskim jest wyrównane do drugiego słowa w pięciowyrazowym zdaniu niemieckim.

Definicja 17. (Produktywność) *Prawdopodobieństwo, że słowo źródłowe f jest tłumaczone na określoną liczbę słów docelowych nazywamy* produktywnością *i oznaczamy przez $n(\phi|f)$, gdzie $\phi = 0, 1, 2, \ldots$.*

Przykład. W kontekście tłumaczenia niemiecko–angielskiego produktywność pewnych wyrazów niemieckich może być określona następująco:

– rzeczownik „Haus" jest niemal zawsze tłumaczony na „house", stąd:

$$n(0|\text{Haus}) \simeq 0$$
$$n(1|\text{Haus}) \simeq 1$$
$$n(2|\text{Haus}) \simeq 0$$

– niemiecki wyraz „zum" będące skrótem od „zu dem" najczęściej tłumaczony jest na „to the", ale możliwe jest również tłumaczenie jedynie na „to" (np. „zur Schule" → „to school"), stąd:

$$n(1|\text{zum}) = \epsilon$$
$$n(2|\text{zum}) = 1 - \epsilon$$

– niemiecka partykuła modulująca „ja" nie ma odpowiednika w języku angielskim, stąd:

$$n(0|\text{ja}) \simeq 1$$

Definicja 18. (Wstawienie NULL) *W sytuacji gdy pewien wyraz zdania docelowego nie jest odpowiednikiem żadnego wyrazu w zdaniu źródłowym, przyjmuje się, że taki wyraz docelowy jest odpowiednikiem wyrazu specjalnego* NULL.

Zakłada się, że wyraz specjalny NULL może wystąpić po każdym wyrazie zdania źródłowego z prawdopodobieństwem p_1. Prawdopodobieństwo, że wyraz NULL nie wystąpi po danym wyrazie zdania źródłowego oznaczamy przez $p_0 = 1 - p_1$.

Przykład. Rysunek 2.1 przedstawia parę zdań i ich wyrównanie, w którym angielski wyraz „do" nie został wyrównany do żadnego wyrazu źródłowego. Operacja wstawienia wyrazu specjalnego NULL pozwala zinterpretować wyraz „do" jako wyrównanego do wyrazu NULL w zdaniu źródłowym „ich NULL gehe ja nicht zum Haus".

Definicja 19. (Dystorsja) *Prawdopodobieństwo, że słowo znajdujące się na pozycji j w zdaniu docelowym jest tłumaczeniem słowa znajdujące się na pozycji i w zdaniu źródłowym nazywamy* dystorsją *i oznaczamy przez $d(j|i, l_e, l_f)$.*

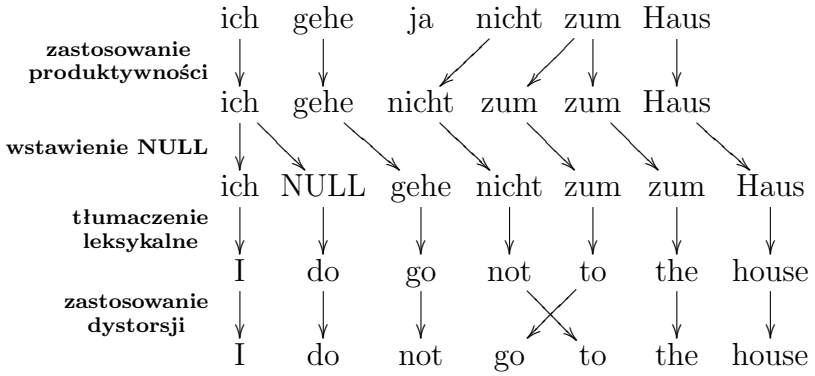
Dystorsja może być traktowana jako „odwrócone" prawdopodobieństwo wyrównania. Prawdopodobieństwo wyrównania „działa" w tym samym kierunku co funkcja wyrównania — „przewiduje" pozycję wyrazów w zdaniu źródłowym na podstawie pozycji wyrazów w zdaniu docelowym. Dystorsja „działa" w kierunku tłumaczenia — „przewiduje" pozycję wyrazów w zdaniu docelowym na podstawie pozycji wyrazów w zdaniu źródłowym.

Definicja 20. (IBM Model 3) *Prawdopodobieństwo, że zdanie $\boldsymbol{e}$ jest tłumaczeniem zdania $\boldsymbol{f}$, określa się następująco:*

$$\begin{aligned} p(\boldsymbol{e}|\boldsymbol{f}) &= p(\boldsymbol{e}, a|\boldsymbol{f}) \\ &= \sum_{a(1)=0}^{l_f} \cdots \sum_{a(l_e)=0}^{l_f} \prod_{j=1}^{l_e} \binom{l_e - \phi_0}{\phi_0} p_1^{\phi_0} p_0^{l_e - 2\phi_0} \prod_{i=1}^{l_f} \phi_i! n(\phi_i|e_i) \\ &\quad \times \prod_{j=1}^{l_e} t(e_j|f_{a(j)}) d(j|a(j), l_e, l_f) \end{aligned}$$

gdzie ϕ_0 to ilość wstawionych wyrazów specjalnych NULL.

Przykład. Wprowadzenie operacji wstawienia wyrazu specjalnego NULL i produktywności powoduje, że tłumaczenie w IBM Model 3 można postrzegać jako czterofazowy proces.



W poszczególnych fazach:

1. Stosowana jest produktywność każdego z wyrazów źródłowych np. „zum" zostaje zduplikowane z powodu dużego prawdopodobieństwa $n(2|\text{zum})$, a „ja" usunięte — z powodu bliskiego jedynce prawdopodobieństwu $n(0|\text{ja})$ (patrz: przykład ilustrujący produktywność, strona 21).
2. Następuje wstawienie wyrazu specjalnego NULL. Po „ich" uwzględnione zostaje prawdopodobieństwo p_1, a na przykład po „nicht" uwzględnia się prawdopodobieństwo $p_0 = 1 - p_1$.
3. Podobnie jak w IBM Model 1 następuje tłumaczenie leksykalne, np. „nicht" tłumaczone jest na „not" z prawdopodobieństwem $t(\text{not}|\text{nicht})$.
4. Dystorsja stosowana jest podobnie jak prawdopodobieństwo wyrównania w IBM Model 2, jednak w tym przypadku prawdopodobieństwo pozycji wyrazu w zdaniu docelowym zależy od od pozycji wyrazu, którego jest on tłumaczeniem, w zdaniu źródłowym. Np. prawdopodobieństwo umiejscowienia angielskiego wyrazu „go" na czwartej pozycji w siedmiowyrazowym zdaniu docelowym, jako tłumaczenie niemieckiego wyrazu „gehe", który znajduje się na drugiej pozycji w zdaniu źródłowym, dane jest prawdopodobieństwem $d(4|2, 7, 6)$.

□

Formuła matematyczna określająca IBM Model 3 stanowi połączenie produktywności, tłumaczenia leksykalnego i dystorsji. Każdy wyraz źródłowy f_i generuje ϕ_i wyrazów docelowych z prawdopodobieństwem $n(\phi_i|f_i)$. Ilość wstawionych wyrazów specjalnych NULL ϕ_0 zależy więc od ilości wyrazów docelowych generowanych przez wyrazy źródłowe. Mamy $\sum_{i=1}^{l_j} \phi_i = l_e - \phi_0$ wyrazów docelowych wygenerowanych ze zdania źródłowego, a więc maksymalnie $l_e - \phi_0$ wstawionych wyrazów NULL.

Prawdopodobieństwo wygenerowania ϕ_0 wyrazów z wyrazu NULL wynosi:

$$p(\phi_0) = \binom{l_e - \phi_0}{\phi_0} p_1^{\phi_0} p_0^{l_e - 2\phi_0}$$

gdzie p_0 to prawdopodobieństwo, że NULL nie został wstawiony po tradycyjnie wygenerowanym wyrazie, a p_1 to prawdopodobieństwo, że wygenerowany został jeden wyraz specjalny NULL. Symbol kombinatoryczny $\binom{l_e - \phi_0}{\phi_0}$ pojawia się w sytuacji, gdy ustalona ilość wyrazów specjalnych NULL może zostać wstawiona po różnych zestawach konwencjonalnie wygenerowanych wyrazów: można wybrać ϕ_0 z $l_e - \phi_0$ pozycji.

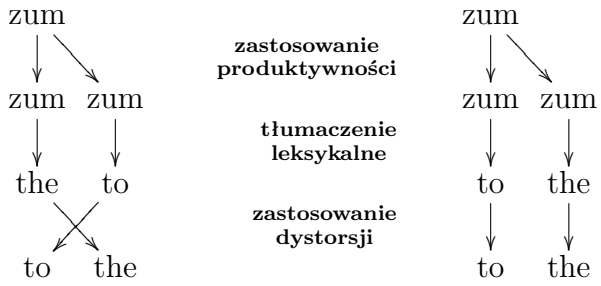
Produktywność z operacją wstawienia wyrazu specjalnego NULL łączy formuła:

$$\binom{l_e - \phi_0}{\phi_0} p_1^{\phi_0} p_0^{l_e - 2\phi_0} \prod_{i=1}^{l_f} \phi_i! n(\phi_i | e_i)$$

Dla każdego wyrazu źródłowego obliczona zostaje produktywność dana w postaci prawdopodobieństwa $n(\phi_i|e_i)$, z uwzględnieniem czynnika $\phi_i!$, który pojawia się ze względu na możliwość wygenerowania tego samego tłumaczenia w różny sposób.

Przykład. Niemieckie „zum" tłumaczy się zwykle na angielski jako „to the". W IBM Model 3 na skutek swojej produktywności „zum" zostaje potraktowany jako dwa wyrazy („zum zum"), z których każdy może zostać przetłumaczony na „to" lub „the".

Możliwe jest więc otrzymanie takiego samego tłumaczenia z tym samym wyrównaniem na dwa różne sposoby.



Definicja 21. (Cept) *Grupę złożoną z wyrazu zdania źródłowego f oraz wyrównanych do niego wyrazów zdania docelowego, nazywamy* cept.

Cept oznaczać będziemy przez π_i, gdzie $i = 1, 2, \ldots, l_f$.

Przykład. Wyrównanie pary zdań przedstawione na rysunku 2.1 pozwala zidentyfikować pięć grup cept:

	f	e
π_1	ich	i
π_2	gehe	go
π_3	nicht	not
π_4	zum	to, the
π_5	Haus	house

Definicja 22. (Operator SWP) *Operator mapujący cept π_i na pozycję (w zdaniu źródłowym) wyrazu źródłowego składającego się na cept π_i nazywamy* operatorem SWP *i oznaczamy przez $[i]$.*

Przykład. Dla wyrównania pary zdań przedstawionego na rysunku 2.1 mamy: $[1] = 1$, $[2] = 2$, $[3] = 4$, $[4] = 5$, $[5] = 6$.

Definicja 23. (Środek cept) *Cechę górną średniej pozycji wyrazów docelowych składających się na cept π_i nazywamy* środkiem cept *i oznaczamy przez $\odot_i$.*

Przykład. Dla wyrównania pary zdań przedstawionego na rysunku 2.1 mamy: $\odot_1 = 1$, $\odot_2 = 4$, $\odot_3 = 3$, $\odot_4 = 6$, $\odot_5 = 7$.

Definicja 24. (Dystorsja względna) *Dla:*

1. *wyrazów wyrównanych do wyrazu specjalnego NULL przyjmuje się, że są one rozmieszczone równomiernie.*
2. *pierwszego wyrazu docelowego w cept π_i* dystorsja względna *jest to prawdopodobieństwo $d_1(j - \odot_{i-1})$ umieszczenia wyrazu docelowego na pozycji j względem środka poprzedniego cept $\odot_i$.*
3. *kolejnych wyrazów docelowych w cept π_i* dystorsja względna *jest to prawdopodobieństwo $d_{>1}(j - \pi_{i,k-1})$ umieszczenia wyrazu docelowego na pozycji j względem pozycji $k-1$-ego wyrazu z cept π_i (oznaczonej przez $\pi_{i,k-1}$).*

Przykład. Dla wyrównania pary zdań przedstawionego na rysunku 2.1 mamy:

j	1	2	3	4	5	6	7
e_j	i	do	not	go	to	the	house
w cept $\pi_{i,k}$	$\pi_{1,0}$	$\pi_{0,0}$	$\pi_{3,0}$	$\pi_{2,0}$	$\pi_{4,0}$	$\pi_{4,1}$	$\pi_{5,0}$
$\odot_{i-1}$	0	–	4	1	3	–	6
$j - \odot_{i-1}$	+1	–	−1	+3	+2	–	+1
dystorsja	$d_1(+1)$	1	$d_1(-1)$	$d_1(+3)$	$d_1(+2)$	$d_{>1}(1)$	$d_1(+1)$

Definicja 25. (IBM Model 4) *Modyfikację IBM Model 3, w której w miejsce dystorsji używa się dystorsji względnej zależnej od klasy słów źródłowych i docelowych, a więc:*

$$d_1(j - \odot_{[i-1]} | \mathcal{A}(f_{[i-1]}), \mathcal{B}(e_j))$$
$$d_{>1}(j - \pi_{i,k-1} | \mathcal{B}(e_j))$$

gdzie $\mathcal{A}(f)$ i $\mathcal{B}(e)$ są funkcjami przydzielającymi wyrazy języka źródłowego i docelowego do pewnej liczby klas, nazywamy IBM Model 4.

W praktyce obserwuje się, że kolejność pewnych słów w zdaniu niemal zawsze podlega zmianie, podczas gdy inne słowa niemal zawsze pozostają w porządku zdania źródłowego. Przykładem „obowiązkowej" zmiany kolejności wyrazów może być inwersja przymiotnika i rzeczownika podczas tłumaczenia z języka francuskiego na angielski: „affaires extérieur" tłumaczy się na „external affairs".

Ze względu na tego typu zjawiska naturalnym wydaje się uzależnienie dystorsji od wyrazów e_j i $f_{[i-1]}$:

$$d_1(j - \odot_{[i-1]} | f_{[i-1]}, e_j)$$
$$d_{>1}(j - \pi_{i,k-1} | e_j)$$

Niestety realistyczne oszacowanie tych rozkładów prawdopodobieństw jest możliwe jedynie przy założeniu dostępności korpusu o rozmiarach znacznie większych niż obecnie dostępnych. Z tego powodu uzależnia się dystorsję względną od *klas wyrazów*. Wprowadzenie *klas wyrazów* redukuje słownictwo danego języka, a przez to zwiększa szanse na zebranie miarodajnych statystyk z dostępnego korpusu.

Klasy wyrazów nie są ściśle zdefiniowane. W skrajnym przypadku może to być zupełnie arbitralny podział. Inną możliwością jest pogrupowanie wyrazów ze względu na część mowy lub funkcję, jaką pełnią w zdaniu.

Definicja 26. (IBM Model 5) *Modyfikację IBM Model 4, w której dystorsję względną określono następująco:*

$$d_1(v_j | \mathcal{B}(e_j), v_{\odot_{i-1}}, v_{max})$$
$$d_{>1}(v_j - v_{\pi_{i,k-1}} | \mathcal{B}(e_j), v_{max'})$$

gdzie:

- v_j *to ilość pozycji w zdaniu docelowym w przedziale* $[1, j]$*, które nie zostały jeszcze „obsadzone" żadnym wyrazem docelowym,*
- v_{max} *to maksymalna liczba „nieobsadzonych" pozycji w zdaniu docelowym,*
- $v_{max'}$ *to maksymalna liczba „nieobsadzonych" pozycji w zdaniu docelowym, obliczana przy założeniu, że nowe wyrazy mogą być umieszczane jedynie za umieszczonymi już wyrazami,*

nazywamy IBM Model 5.

Przykład. Możliwe jest, że w trakcie trenowania modelu niezerowe prawdopodobieństwo zostanie przypisane do wyrównań, w których więcej niż jeden wyraz docelowy zajmować będzie tą samą pozycję. Sytuacja taka jest niedopuszczalna ze względów praktycznych.

IBM Model 5 określa dystorsję w oparciu o jeszcze „nieobsadzone" pozycje w zdaniu docelowym, w przeciwieństwie do IBM Model 3 i 4, które odwołują się do pozycji w zdaniu docelowym niezależnie od tego, czy jest ona zajęta czy nie. W ten sposób IBM Model 5 przeciwdziała marnotrawieniu części masy prawdopodobieństwa na niedopuszczalne w praktyce rezultaty.

Dla wyrównania pary zdań przedstawionego na rysunku 2.1 mamy:

cept		„nieobsadzone" pozycje							parametry dla d_1			
$f_{[i]}$	$\pi_{i,k}$	v_1	v_2	v_3	v_4	v_5	v_6	v_7	j	v_j	v_{max}	$v_{\odot_{i-1}}$
		I	do	not	go	to	the	house				
NULL	$\pi_{0,1}$	1	**2**	3	4	5	6	7	2	-	-	-
ich	$\pi_{1,1}$	**1**	-	2	3	4	5	6	1	1	6	0
gehe	$\pi_{2,1}$	-	-	1	**2**	3	4	5	4	2	5	0
nicht	$\pi_{3,1}$	-	-	**1**	-	2	3	4	3	1	4	1
zum	$\pi_{4,1}$	-	-	-	-	**1**	2	-	5	1	2	0
	$\pi_{4,2}$	-	-	-	-	-	**1**	2	6	-	-	-
Haus	$\pi_{5,1}$	-	-	-	-	-	-	**1**	7	1	1	0

Zdanie angielskie ma w tym przypadku długość 7. Wyraz specjalny NULL i wyrazy niemieckie od $f_{[1]}$ do $f_{[5]}$ generują wyrazy angielskie i umieszczają je na „nieobsadzonych" pozycjach:

- wyraz specjalny NULL generuje angielski wyraz „do" i umieszcza go na pozycji 2; dystorsja wyrazów generowanych przez NULL, podobnie jak w IBM Model 3 i 4, ma rozkład równomierny,
- niemiecki wyraz $f_[1]$ („ich") generuje angielski wyraz „I" na pozycji 1; od początku zdania do tej pozycji włącznie była jedna „nieobsadzona" pozycja; po obsadzeniu tej pozycji jest jeszcze sześć „nieobsadzonych",
- $f_[2]$ umieszcza angielski wyraz „go" na pozycji 4; pozostało pięć „nieobsadzonych" pozycji,
- $f_[3]$ umieszcza angielski wyraz „not" na pozycji 3; ponieważ poprzedni cept został przetłumaczony niezgodnie z kolejnością wyrazów w zdaniu źródłowym, ilość „nieobsadzonych" pozycji $v_{\odot_{i-1}}$ w środku poprzedniego cept wynosi 1 (środkiem $\odot_{i-1}$ jest 4),
- $f_[4]$ generuje dwa angielskie wyrazy: „to" (e_5) i „the" (e_6); ponieważ e_5 musi zostać umieszczony przez e_6, więc pierwsze słowo tworzące ten cept nie może zostać umieszczone na pozycji 7 (ostatnia „nieobsadzona" pozycja)
- wyraz wygenerowany przez $f_[5]$ wypełnia ostatnią „nieobsadzoną" pozycję.

2.3.2. Algorytmy trenujące EM

Zadaniem algorytmu EM jest wytrenowanie modelu, a więc określenie wartości zmiennych używanych w modelu na podstawie pewnych danych treningowych.

Algorytm EM polega na:

1. zainicjowaniu modelu (np. losowymi wartościami),
2. zastosowaniu modelu do danych,
3. dostosowaniu modelu do danych,
4. iteracji kroków 2 i 3 aż do osiągnięcia zbieżności.

Algorytm EM dla IBM Model 1

Aby zastosować model do danych należy obliczyć prawdopodobieństwa różnych wyrównań dla danej pary zdań (źródłowego i docelowego).

$$p(a|\mathbf{e},\mathbf{f}) = \frac{p(\mathbf{e},a|\mathbf{f})}{p(\mathbf{e}|\mathbf{f})}$$

Prawdopodobieństwo przetłumaczenia zdania f jako e wynosi:

$$\begin{aligned}
p(\mathbf{e}|\mathbf{f}) &= \sum_a p(\mathbf{e},a|\mathbf{f}) \\
&= \sum_{a(1)=0}^{l_f} \cdots \sum_{a(l_e)=0}^{l_f} p(\mathbf{e},a|\mathbf{f}) \\
&= \sum_{a(1)=0}^{l_f} \cdots \sum_{a(l_e)=0}^{l_f} \frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} t(e_j|f_{a(j)}) \\
&= \frac{\epsilon}{(l_f+1)^{l_e}} \sum_{a(1)=0}^{l_f} \cdots \sum_{a(l_e)=0}^{l_f} \prod_{j=1}^{l_e} t(e_j|f_{a(j)}) \\
&= \frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} \sum_{i=0}^{l_f} t(e_j|f_i)
\end{aligned}$$

Tak więc prawdopodobieństwo wyrównania dla pary zdań wynosi:

$$\begin{aligned}
p(a|\mathbf{e},\mathbf{f}) &= \frac{p(\mathbf{e},a|\mathbf{f})}{p(\mathbf{e}|\mathbf{f})} \\
&= \frac{\frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} t(e_j|f_{a(j)})}{\frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} \sum_{i=0}^{l_f} t(e_j|f_i)} \\
&= \prod_{j=1}^{l_e} \frac{t(e_j|f_{a(j)})}{\sum_{i=0}^{l_f} t(e_j|f_i)}
\end{aligned}$$

Aby dostosować model do danych należy zliczyć, jak często dane słowo docelowe jest tłumaczeniem pewnego słowa źródłowego przy uwzględnieniu prawdopodobieństw różnych wyrównań zdań źródłowego i docelowego.

$$c(e|f;\mathbf{e},\mathbf{f}) = \sum_a p(a|\mathbf{e},\mathbf{f}) \sum_{j=0}^{l_e} \delta(e,e_j)\delta(f,f_{a(j)})$$

Przez $\delta(x,y)$ oznaczono symbol Kroneckera.

W wyniku rozwinięcia $p(a|\mathbf{e},\mathbf{f})$ otrzymujemy:

$$c(e|f;\mathbf{e},\mathbf{f}) = \frac{t(e|f)}{\sum_{i=0}^{l_f} t(e_j|f_i)} \sum_{j=0}^{l_e} \delta(e,e_j) \sum_{i=0}^{l_f} \delta(f,f_{a(j)})$$

Prawdopodobieństwo tłumaczenia można szacować przy użyciu funkcji częstości $c(e|f;\mathbf{e},\mathbf{f})$ w sposób następujący:

$$t(e|f;\mathbf{e},\mathbf{f}) = \frac{\sum_{(\mathbf{e},\mathbf{f})} c(e|f;\mathbf{e},\mathbf{f})}{\sum_e \sum_{(\mathbf{e},\mathbf{f})} c(e|f;\mathbf{e},\mathbf{f})}$$

Algorytm EM dla IBM Model 2

Prawdopodobieństwo przetłumaczenia zdania f jako e wynosi:

$$\begin{aligned} p(\mathbf{e}|\mathbf{f}) &= \sum_a p(\mathbf{e},a|\mathbf{f}) \\ &= \epsilon \sum_{a(1)=0}^{l_f} \cdots \sum_{a(l_e)=0}^{l_f} \prod_{j=1}^{l_e} t(e_j|f_{a(j)}) a(a(j)|j,l_e,l_f) \\ &= \epsilon \prod_{j=1}^{l_e} \sum_{i=0}^{l_f} t(e_j|f_{a(j)}) a(a(j)|j,l_e,l_f) \end{aligned}$$

Częstości dla tłumaczeń leksykalnych dane są wzorem:

$$c(e|f;\mathbf{e},\mathbf{f}) = \sum_{j=1}^{l_e} \sum_{i=0}^{l_f} \frac{t(e|f)a(a(j)|j,l_e,l_f)\delta(e,e_j)\delta(f,f_i)}{\sum_{i'=0}^{l_f} t(e|f_{i'})a(i'|j,l_e,l_f))}$$

Częstości dla wyrównania dane są wzorem:

$$c(i|j,l_e,l_f;\mathbf{e},\mathbf{f}) = \frac{t(e_j|f_i)a(a(j)|j,l_e,l_f)}{\sum_{i'=0}^{l_f} t(e_j|f_{i'})a(i'|j,l_e,l_f))}$$

W ramach inicjalizacji $t(e|f)$ i $a(i|j,l_e,l_f)$ używa się wyników uzyskanych z iteracji algorytmu EM dla IBM Model 1.

Próbkowanie przestrzeni wyrównań dla IBM Model 3 i wyższych

Koszt jawnego przeanalizowania wszystkich możliwych wyrównań na potrzeby wytrenowania modelu tłumaczenia rośnie wykładniczo względem długości zdania. Dlatego konieczne jest próbkowanie przestrzeni wyrównań polegające na analizowaniu jedynie tych wyrównań, które stanowią większość masy prawdopodobieństwa.

Procedura treningowa składa się z klasycznego algorytmu EM rozszerzonego o kroki związane z próbkowaniem przestrzeni wyrównań:

1. znalezienie najbardziej prawdopodobnego wyrównania,
2. wygenerowanie pewnych wariacji tego wyrównania.

Najbardziej prawdopodobne wyrównanie znajduje się poprzez optymalizację wyrównania obliczonego z użyciem IBM Model 2 metodą gradientu prostego.

Dla zminimalizowania niebezpieczeństwa znalezienia jedynie lokalnego maksimum jako punktu startowego dla procedury optymalizacyjnej używa się najlepszego wyrównania z IBM Model 2 obliczonego przy założeniu jednego punktu stałego w wyrównaniu $a(j) = i$.

Generowanie wariacji najlepszego wyrównania polega na utworzeniu wyrównań spełniających jeden z poniższych warunków:

– dwa wyrównania a_1 i a_2 różnią się jedynie w obrębie jednego słowa j:

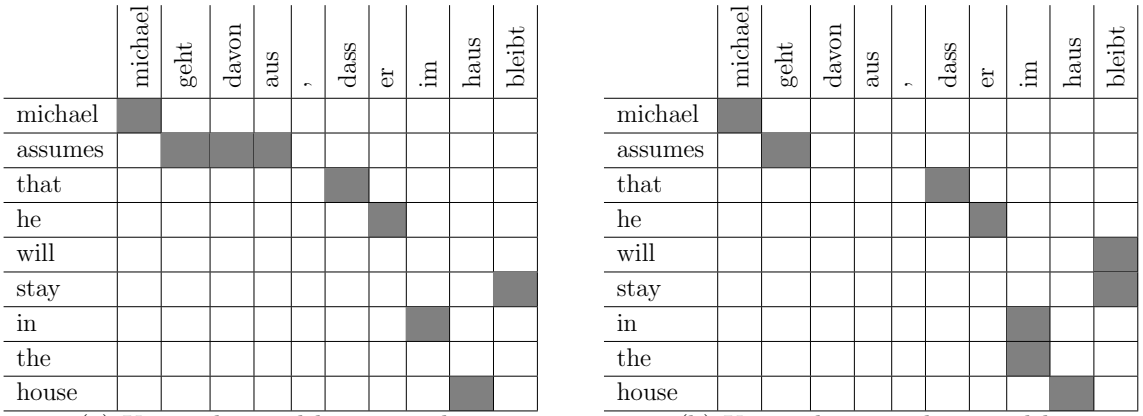
$$\exists j : a_1(j) \neq a_2(j), \forall j' \neq j : a_1(j') = a_2(j')$$

– dwa wyrównania a_1 i a_2 różnią się jedynie w obrębie dwóch słów, dla których wyrównania zostały między sobą zamienione:

$$\begin{aligned} \exists j_1, j_2 \ : \ & j_1 \neq j_2, \\ & a_1(j_1) = a_2(j_2), a_1(j_2) = a_2(j_1), a_2(j_2) \neq a_2(j_1), \\ & \forall j' \neq j_1, j_2 : a_1(j') = a_2(j') \end{aligned}$$

2.3.3. Algorytmy wyrównywania z dokładnością co do wyrazu

Produktem ubocznym tłumaczenia statystycznego w oparciu o IBM Model jest ustanowienie wyrównania z dokładnością co do wyrazu dla każdej pary zdań (źródłowego i docelowego). W trakcie trenowania modelu algorytmem EM, spośród rozważanych wyrównań, wskutek iteracji algorytmu wyłania się najbardziej prawdopodobne wyrównanie, zwane także *wyrównaniem Viterbi'ego*.



(a) Kierunek angielsko–niemiecki (b) Kierunek niemiecko–angielski

Rysunek 2.2. Graficzna reprezentacja wyrównania zdania angielskiego i niemieckiego z użyciem IBM Model.

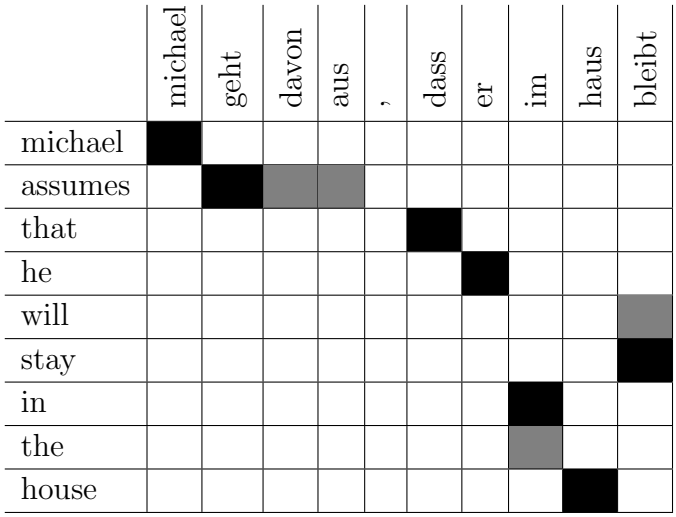
Aby uzyskać wyrównanie pary zdań z dokładnością do wyrazu należy zmodyfikować implementację algorytmu trenującego, tak aby po zakończeniu procedury treningowej wypisane zostało wyrównanie Viterbi'ego.

Ponieważ IBM Model generują wyrazy docelowe e_j z wyrównanych od nich wyrazów źródłowych $f_{a(j)}$ z prawdopodobieństwem $t(e_j|f_{a(j)}i)$, każdy wyraz docelowy może zostać wyrównany jedynie do jednego słowa źródłowego (lub do wyrazu specjalnego NULL).

Przykład. Rysunek 2.2(a) pokazuje wyrównanie, jakie można uzyskać w wyniku trenowania IBM Model w kierunku z angielskiego na niemiecki. Przykład ten pokazuje, że można wygenerować niemiecką frazę „geht davon aus" z angielskiego wyrazu „assumes", jednak jedynie jeden z dwóch angielski wyrazów: „will" lub „stay", może wygenerować niemiecki czasownik „bleibt".

Dla odwrotnego kierunku tłumaczenia tj. z niemieckiego na angielski, możliwe jest wygenerowanie angielskiego „will stay" z niemieckiego „bleibt", ale nie jest możliwe wygenerowanie angielskiego „assumes" ze wszystkich trzech wyrazów niemieckiej frazy „geht davon aus". Zilustrowano to na rysunku 2.2(b).

Aby uzyskać efekt wyrównania jeden do wielu w kierunku tłumaczenia i odwrotnym stosuje się operację *symetryzacji* polegającą na połączeniu wyrównań otrzymanych z trenowania modelu dla tłumaczenia z języka źródłowego na docelowy i z trenowania modelu w kierunku z języka źródłowego na docelowy.



Rysunek 2.3. Graficzna reprezentacja przecięcia (kolor czarny) i sumy (kolor czarny i szary) wyrównań dla obu kierunków tłumaczenia zdania angielskiego i niemieckiego z użyciem IBM Model.

Połączenie wyrównań może być zdefiniowane np. jako ich przecięcie lub suma. Przecięcie gwarantuje wysoką jakoś wyrównania, gdyż akceptowane są tylko punkty, które zostały wyznaczone w obu kierunkach, a więc takie co do których można mieć „pewność", że są poprawne. Tymczasem w wyniku sumy otrzymuje się „gęstsze" wyrównania, a więc takie, w których nie ma wyrazów niewyrównanych. Niestety wyrównanie w tym przypadku może być obarczone pewnym błędem.

Przykład. Na rysunku 2.3 przedstawiono wynik przecięcia i sumy wyrównań z rysunku 2.2.

2.4. Algorytmy tłumaczenia regułowego z elementami statystycznymi

Heidi Fox w roku 2002 empirycznie zbadała, jak istniejące modele tłumaczenia statystycznego radzą sobie z różnicami w składni jeżyka źródłowego i docelowego [8]. Badanie to pokazało, że błędy na etapie parsowania znacznie zakłócają funkcjonowanie zazwyczaj dobrze działających modeli.

Ponadto, modele te nie są w stanie uchwycić wielu powszechnych wzorców tłumaczenia wynikających z różnicy w składni języka źródłowego i docelowego. np. transformacji struktury SVO na SOV. (Struktura SVO jest charakterystyczna dla ponad 75% języków [5], w tym także dla języka polskiego, i oznacza zdanie, w którym podmiot występuje przed orzeczeniem, a dopełnienie występuje na końcu. O strukturze SOV mówimy, gdy podmiot występuje przed dopełnieniem, a orzeczenie — na końcu. Występuje ona w języku arabskim, węgierskim, hindi i innych).

W rozdziałach 2.4.1 i 2.4.2 opisane zostały algorytmy, których celem jest automatyczne pozyskanie wzorców tłumaczenia radzących sobie lepiej z różnicami składniowymi niż modele statystyczne. Algorytmy te analizują struktury danych, które są używane w procesie trenowania modeli statystycznych, rozszerzając je o wyniki analizy składniowej języka docelowego (2.4.1) lub języka źródłowego (2.4.2).

2.4.1. Algorytm rozkładu grafu: zdanie źródłowe — drzewo docelowe

W roku 2004 Michel Galley, Mark Hopkins, Kevin Knight i Daniel Marcu, zaproponowali teorię, która określa semantykę dla wyrównań korpusu równoległego z dokładnością co do wyrazu [9]. Jednocześnie zaproponowali algorytm, który z wyrównanego z dokładnością co do wyrazu korpusu równoległego, wyprowadza motywowane składniowo *reguły transformacji* mające modelować proces tłumaczenia.

Teoria wyrównań z dokładnością co do wyrazu

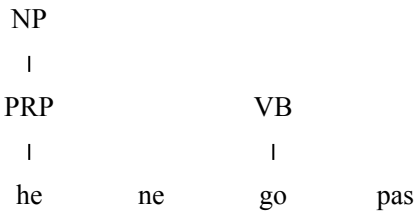
Niech Δ będzie alfabetem.

Definicja 27. (Drzewo symboli) *Drzewem symboli ponad alfabetem Δ nazywamy zakorzenione, skierowane drzewo, którego węzły oznaczone są etykietami będącymi symbolami alfabetu Δ.*

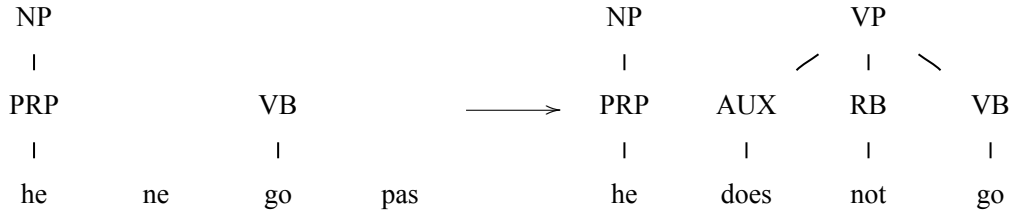
Dane są dwa alfabety: Δ_S — źródłowy i Δ_T — docelowy.

Definicja 28. (Symbol źródłowy (docelowy)) *Symbole alfabetów Δ_S i Δ_T nazywane będą odpowiednio* symbolami źródłowymi *i* docelowymi.

Definicja 29. (Drzewo (poddrzewo) docelowe) *Drzewo symboli ponad alfabetem docelowym nazywać będziemy* drzewem docelowym.
Podgraf drzewa docelowego, który jest drzewem, to poddrzewo docelowe.



Rysunek 2.4. Przykładowa formuła derywacyjna.



Rysunek 2.5. Przykładowy krok derywacji.

Definicja 30. (Formuła derywacyjna) *Ciąg symboli źródłowych i poddrzew docelowych nazywamy* formułą derywacyjną.

Przykład. Na rysunku 2.4 przedstawiono 4-elementowy ciąg złożony z dwóch drzew analizy składniowej dla języka angielskiego (poddrzew docelowych) i dwóch wyrazów języka francuskiego (symboli źródłowych). Ciąg ten jest formułą derywacyjną.

Definicja 31. (Krok derywacji) *Niech S będzie formułą derywacyjną.*

Krokiem derywacji *nazwiemy zastąpienie ciągu $S' \subseteq S$ poddrzewem docelowym T, o następujących właściwościach:*

1. *Każde poddrzewo docelowe, będące elementem ciągu S jest poddrzewem T.*
2. *Drzewo T nie ma żadnych węzłów wspólnych z którymkolwiek poddrzewem docelowym będącym elementem S lecz nie S'.*

Przykład. Na rysunku 2.5 pokazano pewien krok derywacji. Polega on na zastąpieniu 3-elementowego podciągu `ne VB-go pas` formuły derywacyjnej po lewej stronie, jednym drzewem docelowym o liściach `does not go`.

Definicja 32. (Derywacja) *Niech S będzie ciągiem symboli źródłowych, a T drzewem docelowym.*

Ciąg kroków derywacji, które przeprowadzają S w T nazywamy derywacją.

Przykład. Na rysunku 2.6 pokazano trzy różne derywacje przeprowadzające zdanie francuskie w drzewo parsowania zdania angielskiego. Przeprowadzenie zdania w drzewo może odbywać się z różną „prędkością" (w czterech, trzech lub nawet w jednym kroku derywacji). Ponadto krok derywacji nie musi być uzasadniony lingwistycznie, czego przykładem jest np. zastąpienie francuskiego `il` poddrzewem docelowym `RB-not`, które dokonuje się w pierwszym kroku drugiej derywacji.

Definicja 33. (Wyróżnione kroki derywacji) *Niech S będzie ciągiem symboli źródłowych, T drzewem docelowym, a D derywacją przeprowadzającą S w T.*

Dla każdego symbolu źródłowego $s \in S$ określa się $replaced(s, D)$ jako krok derywacji, w którym element s został zastąpiony pewnym poddrzewem docelowym $t' \subset T$.

Dla każdego węzła drzewa docelowego $t \in T$ określa się $created(t, D)$ jako krok derywacji, w którym poddrzewo docelowe $T' \subset T$, takie że $t \in T'$ zastąpiło pewien element w formule derywacyjnej derywacji D.

Definicja 34. (Wyrównanie indukowane przez derywację) *Niech S będzie ciągiem symboli źródłowych, T drzewem docelowym, a D derywacją przeprowadzającą S w T.*

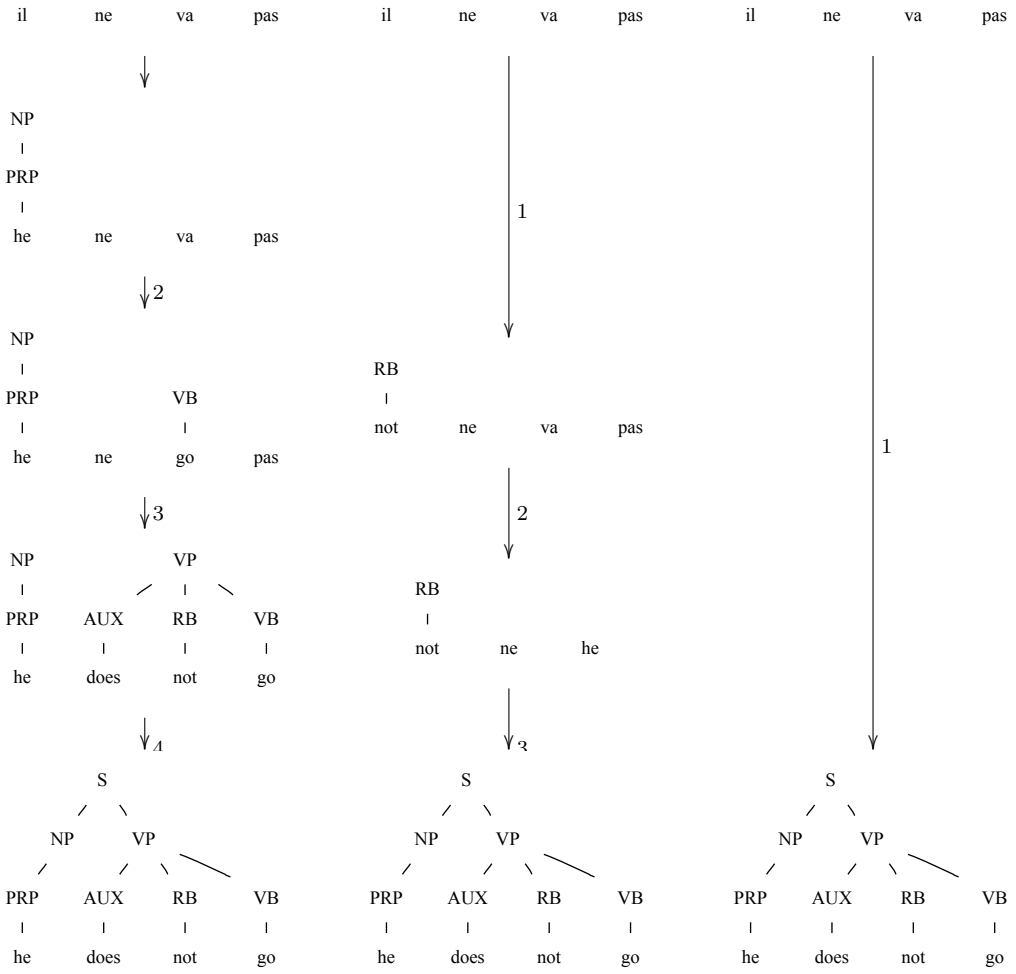
Wyrównaniem A indukowanym przez derywację D *nazywamy relację pomiędzy elementami ciągu źródłowego S, a liśćmi drzewa docelowego T, taką że element $s \in S$ jest w relacji „wyrównany do" z liściem drzewa docelowego $t \in T$ wtedy i tylko wtedy gdy $replaced(s, D) = created(t, D)$.*

Przykład. Na rysunku 2.7 pokazano wyrównania indukowane przez derywacje pokazane na rysunku 2.6.

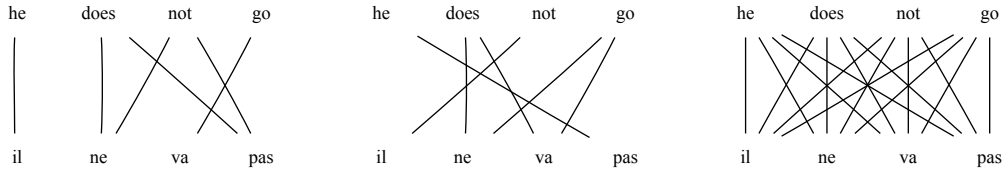
Definicja 35. (Podwyrównanie) *Niech A i A' będą wyrównaniami.*

Wyrównanie $A \subset A'$ jest podwyrównaniem A' wtedy i tylko wtedy gdy elementy będące w relacji A są również w relacji A'.

Przykład. Przedstawione na rysunku 2.7 wyrównanie indukowane przez derywację D_1 jest podwyrównaniem wyrównania indukowanego przez derywację D_3.



Rysunek 2.6. Trzy różne derywacje przeprowadzające zdanie francuskie w drzewo parsowania zdania angielskiego.



D_1 D_2 D_3

Rysunek 2.7. Wyrównania indukowane przez derywacje z rysunku 2.6.

Definicja 36. (Derywacje dopuszczone przez wyrównanie) *Niech S będzie ciągiem symboli źródłowych, T drzewem docelowym, a A wyrównaniem elementów ciągu S względem liści drzewa T.*

Zbiorem derywacji $\delta_A(S,T)$ przeprowadzających ciąg źródłowy S w drzewo docelowe T dopuszczonych przez wyrównanie A nazywamy zbiór tych derywacji D, które indukują pewne wyrównanie A' takie że $A \subset A'$.

Przykład. Derywacja D_3 przedstawiona na rysunku 2.6 jest dopuszczona przez wyrównanie indukowane przez derywację D_1 (przedstawione na rysunku 2.7), gdyż wyrównanie indukowane przez derywację D_1 jest podwyrównaniem wyrównania indukowanego przez derywację D_3.

Definicja 37. (Zbiór reguł) *Niech S będzie ciągiem symboli źródłowych, T drzewem docelowym, a A wyrównaniem elementów ciągu S względem liści drzewa T.*

Zbiorem reguł $\rho_A(S,T)$ nazywać będziemy zbiór złożony z kroków każdej derywacji D takiej że $D \in \delta_A(S,T)$.

Przykład. Kroki derywacji D_1 i D_3 (przedstawionych na rysunku 2.6) należą do zbioru reguł ρ_{A_1}, gdzie A_1 to wyrównanie indukowane przez derywację D_1 (przedstawioną na rysunku 2.7).

Reprezentacja danych

Algorytm zdolny wyznaczyć zbiór $\rho_A(S,T)$ musi przetwarzać dane w postaci uporządkowanej trójki (S,T,A), gdzie S jest ciągiem symboli źródłowych, T drzewem docelowym, a A wyrównaniem elementów ciągu S względem liści drzewa T.

Ciągiem symboli źródłowych S są słowa zdania języka źródłowego. Drzewem docelowym T jest drzewo analizy składniowej zdania w języku docelowym. Wyrównanie A określa odpowiedniość pomiędzy słowami języka źródłowego a docelowego.

Algorytm pozyskiwania reguł

Pozyskanie reguł ze zbioru zbioru $\rho_A(S,T)$ dla danej trójki (S,T,A), gdzie S jest ciągiem symboli źródłowych, T drzewem docelowym, a A wyrównaniem elementów ciągu S względem liści drzewa T, polega na wyznaczeniu *minimalnych podgrafów wiodących.*

Definicja 38. (Rozpięcie węzła n**)** *Rozpięciem węzła n drzewa T nazywamy zbiór $span(n)$ tych symboli źródłowych, które są w relacji A z liśćmi drzewa T osiągalnych z węzła n.*

Definicja 39. (Ciągłe rozpięcie węzła n**)** *Rozpięcie $span(n)$ węzła n jest ciągłe jeśli elementy tego zbioru tworzą podciąg ciągu źródłowego.*

Definicja 40. (Domknięcie rozpięcia węzła n**)** *Najmniejsze ciągle rozpięcie węzła n nazywamy domknięciem rozpięcia węzła n. i oznaczamy przez $closure(span(n))$.*

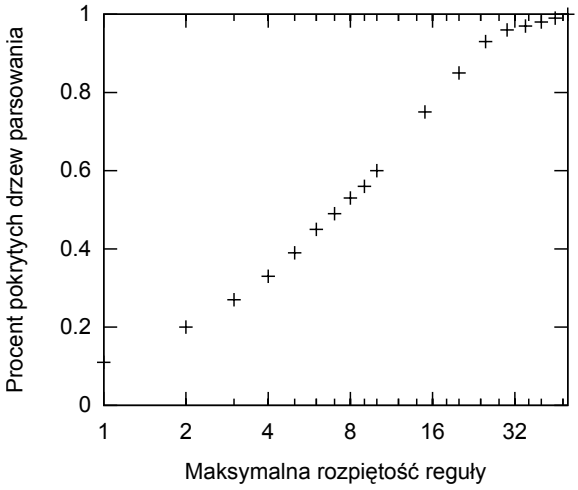
Definicja 41. (Wierzchołek wiodący) *Wierzchołek n drzewa T nazywamy* wiodącym *wtedy i tylko wtedy gdy dla każdego wierzchołka n' drzewa T, nie będącego przodkiem ani potomkiem węzła n, zachodzi $span(n) \cap closure(span(n)) = \emptyset$.*

Definicja 42. (Podgraf wiodący) *Poddrzewo drzewa T takie że korzeń poddrzewa i wszystkie jego liście są wierzchołkami wiodącymi, nazywamy* podgrafem wiodącym.

Definicja 43. (Minimalny podgraf wiodący) *Podgraf wiodący będący podgrafem każdego podgrafu wiodącego o tym samym korzeniu, nazywamy* minimalnym podgrafem wiodącym.

Aby pozyskać reguły z danej trójki (S,T,A), gdzie S jest ciągiem symboli źródłowych, T drzewem docelowym, a A wyrównaniem elementów ciągu S względem liści drzewa T, należy:

1. Wyznaczyć zbiór wierzchołków wiodących.
2. Dla każdego wierzchołka wiodącego n wyznaczyć minimalny podgraf wiodący zakorzeniony w n.
3. Skonwertować każdy minimalny podgrafu wiodącego do kroku derywacji porządkując liście podgrafu wiodącego w kolejności rozpięcia liści. Tak posortowana lista określi element formuły derywacji zastępowany podczas kroku derywacji.



Rysunek 2.8. Zależność ilości drzew parsowania możliwych do pokrycia przez reguły od maksymalnego rozmiaru reguł.

Weryfikacja algorytmu pozyskującego reguły

Opisany w rozdziale 2.4.1 algorytm został poddany weryfikacji polegającej na określeniu w jakim stopniu pozyskane reguły „wyjaśniają" procesy językowe obserwowane w trakcie tłumaczenia.

W eksperymencie użyty został angielsko–chiński korpus równoległy FBIS [2] o rozmiarze około ośmiu milionów słów po stronie angielskiej. Korpus został urównoleglony automatycznie przy użyciu pakietu GIZA++ [22], którego model został wytrenowany wcześniej na większym korpusie (około 150 milionów słów).

Pozyskany zbiór reguł poddano analizie ze względu na rozpiętość reguł. Wykazano, że dobry model zmian składniowych obserwowanych w użytym korpusie musi być w stanie uchwycić zależności występujące pomiędzy wyrazami znacznie od siebie oddalonymi w obrębie danego zdania, co ilustruje wykres na rysunku 2.8. Tym samym zakwestionowano zasadność założenia przyjmowanego we współczesnych statystycznych modelach tłumaczenia, polegającego na dopuszczaniu jedynie lokalnych zmian kolejności.

2. Korpus FBIS oznaczony jest numerem katalogowym Narodowego Instytutu Standardów i Technologii przy Departamencie Handlu USA: LDC2003E14

2.4.2. Regułowo-statystyczny algorytm tłumaczenia automatycznego z języka polskiego

W 2009 roku Marcin Junczys-Dowmunt przedstawił prototyp systemu tłumaczącego Bonsai. Jest to system tłumaczenia statystycznego opartego o składnię dla tłumaczenia polsko–francuskiego [16].

Transducer drzewo–łańcuch

Dane są dwa alfabety: Σ — źródłowy i Δ — docelowy. Niech T_Σ będzie zbiorem uporządkowanych, skończonych, zakorzenionych drzew ponad alfabetem Σ. Liście drzew tych drzew mogą być oznaczone etykietami $X = \{x_1, x_2, x_3, \ldots\}$.

Definicja 44. (xRS) *Transducer M typu xRS to uporządkowana piątka*

$$M = (\Sigma, \Delta, Q, q_0, R)$$

gdzie:

- *Q to skończony zbiór stanów,*
- *$q_0 \in Q$ to wyróżniony stan początkowy,*
- *R to skończony zbiór* ważonych reguł transformacji.

Ważona reguła transformacji $(q, \delta) \xrightarrow{w} rhs$ przekształca, z wagą $w \in \mathbb{R}+$, poddrzewo źródłowe pasującego do wzorca δ, gdy transducer M znajduje się w stanie $q \in Q$, w łańcuch $rhs \in (\Delta \cup (Q \times X))^*$.

Definicja 45. (Zdanie źródłowe) *Łańcuch drzew źródłowych $f \in (Q \times T_\Sigma)^*$ nazywać będziemy zdaniem źródłowym.*

Definicja 46. (Derywacja) *Niech f będzie zdaniem źródłowym, a e łańcuchem docelowym.*

Derywacją nazywamy uporządkowaną trójkę (f, e, h), gdzie $h \in (\mathbb{N} \times R)^*$ to historia derywacji.

Definicja 47. (Koszt derywacji) *Niech (f, e, h) będzie derywacją transducera M.*

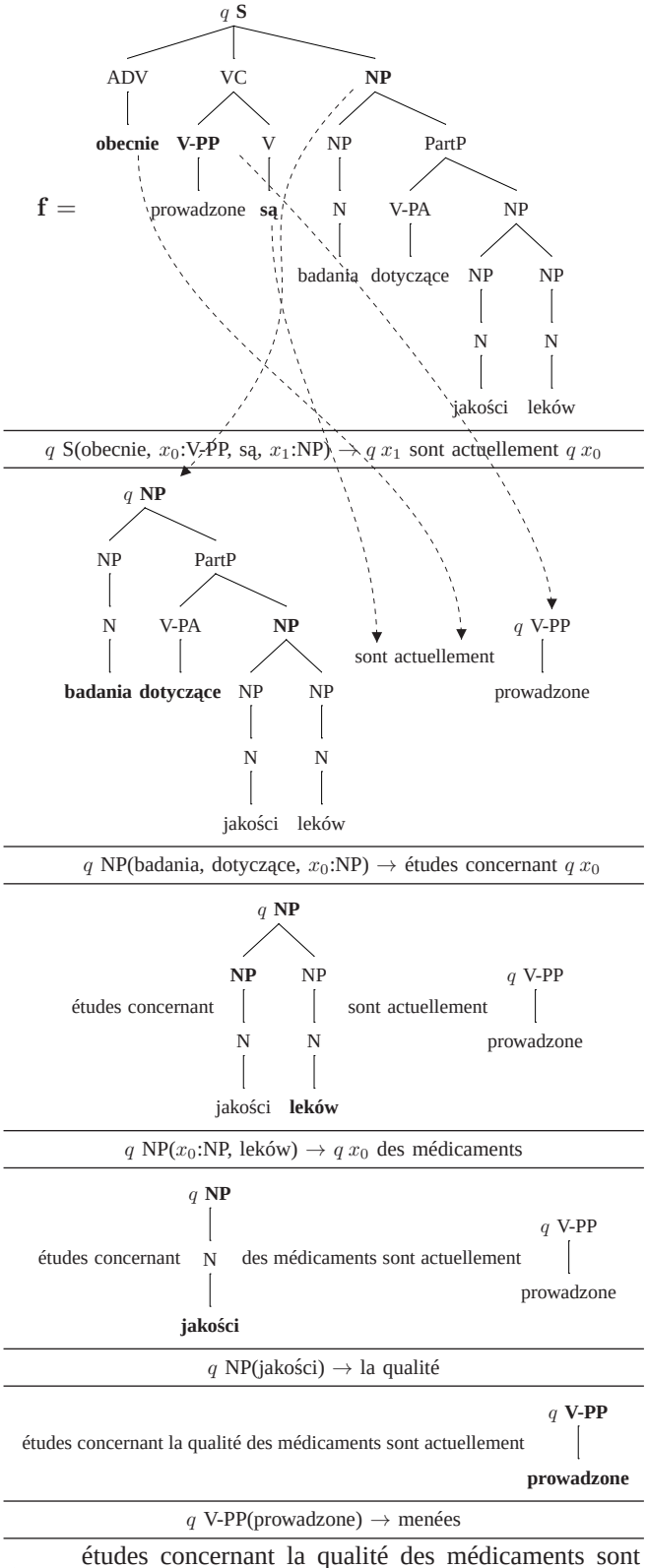
Koszt derywacji w_M, której historia ma długość n określa się następująco:

$$w_M((f, e, h)) = \prod_{i=1}^{n} w_i$$

gdzie w_i to waga skojarzona z regułą transducera stanowiącą i-ty etap derywacji.

Całkowity koszt derywacji W_M, *definiuje się następująco:*

$$W_M((f, e)) = \sum_{(f,e,h) \in LD(M)} w_M((f, e, h))$$



Rysunek 2.9. Przykładowa derywacja obejmująca zamianę kolejności symboli terminalnych i nieterminalnych. (Źródło: [16])

Definicja 48. (Najlepsze tłumaczenie) *Najlepszym tłumaczeniem zdania źródłowego f przy użyciu transducera M nazywamy łańcuch docelowy $\hat{e}$, taki że:*

$$\begin{aligned} \hat{e} &= \underset{e}{argmax}\left(W_M\left((f,e)\right)\right) \\ &\approx \underset{e}{argmax}\left(\underset{(f,e,h)\in LD(M)}{max} w_M\left((f,e,h)\right)\right) \end{aligned}$$

Definicja 49. (Drzewo derywacji) *Niech f będzie zdaniem źródłowym, a M transducerem typu xRS.*

Drzewem G_f derywacji rozpoczynających się w f nazywamy skierowany acykliczny graf, którego wierzchołkami są formuły derywacyjne, a ważonymi krawędziami — produkcje transducera, występujące w historii derywacji h, dla każdego łańcucha docelowego t, takiego że $(f,t,h) \in LD(M)$.

Algorytm tłumaczący

Dla danego zdania źródłowego f i transducera M typu xRS najlepsze tłumaczenie $\hat{e}$ może zostać wyznaczone w oparciu o najlepszą derywację $\hat{h}$:

$$(\hat{e},\hat{h}) = \underset{(e,h)}{argmax}\left(w_M((f,e,h))\right)$$

Znalezienie najlepszej derywacji $\hat{h}$ odpowiada znalezieniu najkrótszej ścieżki w grafie derywacji G_f prowadzącej do pewnego wierzchołka $\hat{e}$.

Przykładowa derywacja obejmująca zamianę kolejności symboli terminalnych i nieterminalnych została pokazana na rysunku 2.9.

Weryfikacja algorytmu tłumaczącego

Do weryfikacji algorytmu użyte zostały pary zdań polsko–francuskich pochodzących z korpusu JRC-Acquis [26]. Dwa tysiące par zdań użytych zostało jako dane testowe. Pozostałe pary — około 1,2 mln, użyte zostały jako dane treningowe.

Korpus treningowy został wyrównany z dokładnością do słowa przy użyciu pakietu Giza++. Strona źródłowa korpusu treningowego został przeanalizowana składniowo z użyciem płytkiego parsera Spejd [25]. Z tak przygotowanego korpus uzyskano około 50 mln unikatowych reguł.

Algorithm	$\bar{t}$ [s]	BLEU	METEOR
Moses Default		0.5906	0.3944
Moses Mert		0.6259	0.4340
Bonsai Greedy	0.05	0.6022	0.3972
Bonsai Astar	1.33	0.6104	0.4123
Bonsai Stack[5]	0.20	0.6140	0.4067
Bonsai Stack[10]	0.35	0.6166	0.4122
Bonsai Stack[20]	0.63	0.6194	0.4133
Bonsai Stack[50]	1.45	0.6234	0.4188
Bonsai Stack[100]	2.74	0.6247	0.4194

Rysunek 2.10. Wyniki automatycznej oceny jakości tłumaczenia pewnych wariantów systemów Moses i Bonsai. (Źródło: [16])

Sprawność algorytmu tłumaczącego została oceniona w sposób automatyczny przy użyciu metryk Blue i Meteor. Otrzymane wyniki zostały porównane z wynikami jakie na tym samym korpusie udało się osiągnąć z użyciem narzędzi do tłumaczenia statystycznego z pakietu Moses [18]. Rysunek 2.10 przedstawia obserwowany średni czas tłumaczenie pojedynczego zdania i wyniki automatycznej oceny tłumaczenia dla różnych konfiguracji systemów Moses i Bonsai.

Rozdział 3

Algorytmy tłumaczenia regułowego oparte o małe korpusy dwujęzyczne

Założenia przyjmowane dla algorytmów opisanych w rozdziale 2 mogą w praktyce okazać się trudne do spełnienia. W przypadku systemów regułowych przeszkodą mogą być ramy czasowe lub ograniczenia ekonomiczne związane z danym projektem. [1] W przypadku systemów statystycznych trudność może polegać na braku odpowiednio dużych zestawów danych treningowych. [2]

Poniżej przedstawione zostaną autorskie algorytmy, które na etapie przygotowania danych wymagają pewnego wkładu pracy ludzkiej, lecz jest to wkład nieporównywalnie mniejszy czasowo niż w klasycznym tłumaczeniu regułowym. W stosunku do klasycznych metod statystycznych proponowane tutaj podejście wymaga znacznie mniejszych zasobów dwujęzycznych [15, 19].

3.1. Warunki wstępne

Poniżej sformułowane i objaśnione zostaną warunki określające środowisko, dla którego zostały zaprojektowane przedstawione dalej algorytmy.

Warunek 1. (Parser języka docelowego) *Dostępność korpusu dwujęzycznego, w którym zdania w języku docelowym zostały oznaczone składniowo.*

Za przyjęciem powyższego założenia przemawiają względy praktyczne. Celem rozwojowym badań prezentowanych w niniejszej pracy jest zastosowanie opracowanych algorytmów w systemie tłumaczenia automatycznego na język polski. Istnieją zasoby językowe oraz skuteczny parser dla języka polskiego [3], co umożliwia dokonanie wiarygodnego oznaczenia składniowego zdań polskich. Sama metodyka badań abstrahuje jednak od własności języków, między którymi odbywa się tłumaczenie, i może być stosowana dla innych języków.

1. Z jak dużym nakładem pracy wiąże się wytworzenie regułowego systemu tłumaczącego można dowiedzieć się na przykładzie systemu POLENG z pracy [12].
2. Problem ten pojawia się szczególnie w przypadku tzw. pi-języków [27].
3. Patrz: opis systemu POLENG w rozdziale 2.2.

Warunek 2. (Wyrównanie co do wyrazu) *Dostępność korpusu dwujęzycznego, w którym dla każdej pary zdań oznaczono odpowiedniość wyrazów pomiędzy zdaniem źródłowym a docelowym.*

Wyrównanie korpusu z dokładnością do wyrazu jest podstawowym źródłem wiedzy, z którego czerpią wszystkie metody przedstawione w rozdziale 2. W przypadku algorytmów statystycznych wyrównanie to konstruowane jest niejawnie. W przypadku algorytmów regułowych wiedza możliwa do pozyskania z wyrównania z dokładnością do słowa zawarta jest w słowniku dostarczonym do systemu tłumaczącego. W proponowanym tutaj podejściu regułowo–statystycznym zakładamy, że dostępny jest korpus dwujęzyczny, w którym oznaczono odpowiedniość wyrazów.

W porównaniu do metody czysto statystycznej nasze podejście ma znacznie mniejsze oczekiwania w stosunku do wielkości korpusu uczącego.

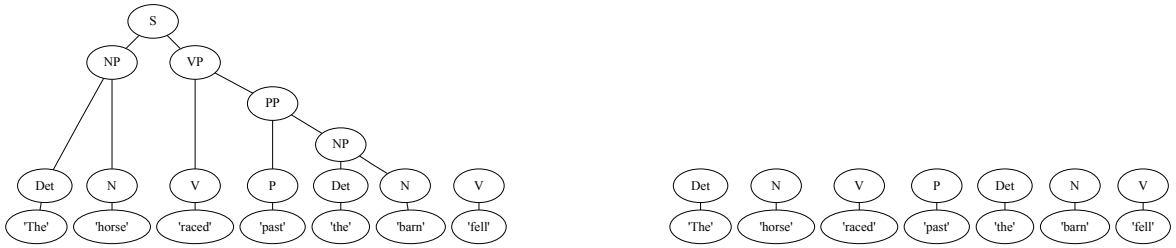
Warunek 3. (Mały korpus) *Dostępność korpusu dwujęzycznego o rozmiarze co najmniej kilkuset par zdań.*

Za przyjęciem powyższego założenia ponownie przemawiają względy praktyczne. Systemy tłumaczenia automatycznego okazują się najbardziej przydatne do tłumaczenia tekstów specjalistycznych (gdzie szczególnie istotne jest poprawne i jednoznaczne tłumaczenie terminów z danej dyscypliny). Dla tekstów z wąskiej dziedziny możliwe jest zazwyczaj zebranie jedynie małego zbioru tekstów dwujęzycznych.

Przyjęcie powyższego warunku 3 ma również uzasadnienie kognitywistyczne. W roku 2000 grupa naukowców z Uniwersytetu Południowej Kalifornii [2] przeprowadziła następujący eksperyment: Grupie ludzi, których językiem ojczystym był język angielski, postawiono zadanie przetłumaczenia kilku zdań z języka tetum [4] na angielski. Ludzie ci nie znali tetum, a jako jedyną pomoc w tłumaczeniu otrzymali przygotowany przez organizatorów eksperymentu niewielki (1102 par zdań) korpus dwujęzyczny. Dysponując jedynie tym zasobem, uczestnicy eksperymentu zdołali sformułować poprawne pod względem treści angielskie tłumaczenie zdań oryginalnie sformułowanych w języku tetum.

Należy zaznaczyć, że dla uczestników tego eksperymentu, z racji ich biegłej znajomości języka angielskiego, dostarczony korpus dwujęzyczny był niejawnie oznaczony składniowo po stronie docelowej, a więc spełniony był warunek 1. Poza tym uczestnicy eksperymentu większość czasu potrzebnego do uzyskania tłumaczenia poświęcili na oznaczenie odpowiedniości wyrazów lub fraz w parach zdań w otrzymanym korpusie (patrz: warunek 2).

4. Tetum to język z rodziny austronezyjskiej, używany głównie w Timorze Wschodnim, gdzie wraz z portugalskim jest językiem urzędowym.



(a) Przykładowy graf

(b) Przykładowy podgraf grafu (a)

Rysunek 3.1. Przykładowy graf zbudowany na podstawie zdania „The horse raced past the barn fell".

3.2. Podstawowe pojęcia z teorii grafów

Definicja 50. (Graf) *Uporządkowaną parę $G = (V, E)$ nazywamy grafem, jeżeli V jest zbiorem skończonym, a E jest zbiorem skończonym nieuporządkowanych par $\{u, v\}$, gdzie $u, v \in V$ i $u \neq v$.*

Elementy zbioru V nazywamy wierzchołkami grafu*, a elementy zbioru E* krawędziami grafu *i stosujemy dla nich oznaczenie (u, v).*

Wierzchołek u sąsiaduje z *v jeżeli istnieje krawędź (u, v).*

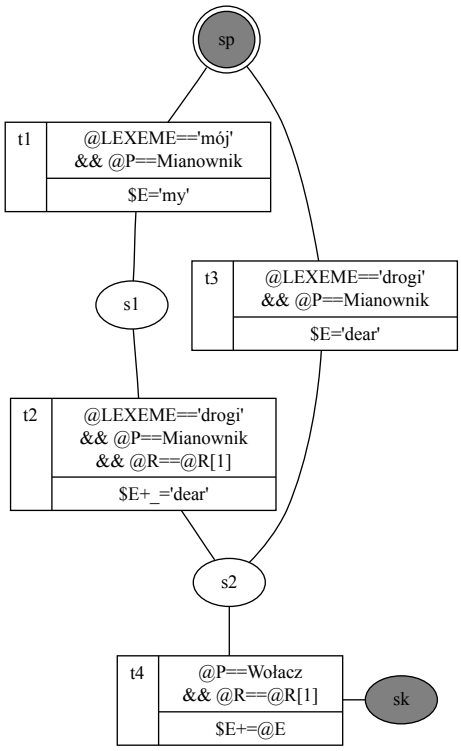
Przykład. Na rysunku 3.1(a) pokazany jest przykład grafu analizy składniowej zdania „The horse raced past the barn fell". Zdanie użyte w tym przykładzie zostało celowo tak zbudowane, aby zakłócić pracę ludzkiego mechanizmu parsującego [4] — pierwszych sześć wyrazów od lewej traktowanych jest jako zdanie, a ostatni wyraz pozostaje bez związku z pozostałymi.

Definicja 51. (Podgraf) *Graf $H = (V', E')$ nazywamy podgrafem grafu $G = (V, E)$, jeżeli $V' \subset V$, a $E' = \{(u, v) \in E : u, v \in V'\}$.*

Przykład. Graf pokazany na rysunku 3.1(b) jest podgrafem grafu z rysunku 3.1(a). Przedstawia on wyniki analizy leksykalnej zdania „The horse raced past the barn fell".

Definicja 52. (Graf indukowany) *Niech $H = (V', E')$ będzie podgrafem $G = (V, E)$. Jeżeli $\forall u, v \in V' : (u, v) \in E' \iff (u, v) \in E$, to H nazywamy grafem indukowanym przez V' i oznaczamy przez $G[V']$.*

Przykład. Graf 3.1(b) jest grafem indukowanym przez zbiór zawierający wierzchołki oznaczone wyrazami zdania „The horse raced past the barn fell" i wierzchołki oznaczone kategorią gramatyczną tych wyrazów.



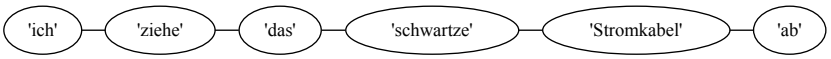
Rysunek 3.2. Przykładowa sieć przejścia

Definicja 53. (Ścieżka) *Niech $G = (V, E)$ będzie grafem.* Ścieżką *nazywamy taki ciąg wierzchołków $\langle v_0, v_1, \ldots, v_n \rangle$, że $\forall i : v_i \in V$ oraz $\forall 1 \leqslant i \leqslant n : (v_{i-1}, v_i) \in E$. Jeżeli ponadto $\forall i \forall j \neq i : v_i \neq v_j$, to ścieżkę nazywamy* ścieżką prostą.

Wierzchołek v_n jest osiągalny z *wierzchołka v_0 jeżeli istnieje ścieżka $\langle v_0, v_1, \ldots, v_n \rangle$.*

Przykład. Na rysunku 3.2 pokazana została tak zwana *sieć przejścia*. Jest to graf, w którym spośród wierzchołków wyróżnia się dwa — wierzchołek początkowy i końcowy, a którego krawędzie etykietuje się tak zwanym warunkami i akcjami. Sieć przejścia stosuje się w ten sposób, że dla danej ścieżki prostej pomiędzy wierzchołkiem początkowym i końcowym, jeżeli w pewnych okolicznościach zostaną spełnione wszystkie warunki przypisane do krawędzi pomiędzy wierzchołkami tej ścieżki, to można wykonać akcje przypisane do tych krawędzi [14, 13].

W grafie na rysunku 3.2 istnieje ścieżka $\langle sp, s1, s2, sk \rangle$. Gdyby w tekście polskim pojawiła się fraza spełniająca warunki przypisane do krawędzi $t1$, $t2$ i $t6$ np. „Moja Droga Anno", wtedy w efekcie podjęcia akcji przypisanych do krawędzi, mogłaby ona zostać przetłumaczona na „My Dear Ann".



Rysunek 3.3. Łańcuch zbudowany na podstawie zdania „Ich ziehe das szwarze Stromkabel ab".

Definicja 54. (Spójna składowa) *Niech G będzie grafem.* Spójnymi składowymi *grafu G nazywamy grafy indukowane przez klasy abstrakcji wierzchołków w relacji „jest osiągalny z".*

Graf, który ma tylko jedną spójną składową, nazywamy grafem spójnym.

Przykład. Graf z rysunku 3.1(a) nie jest spójny, gdyż ma dwie spójne składowe. Pierwsza składowa to graf indukowany przez wierzchołki `V` i `fell`. Druga składowa to graf indukowany przez pozostałe wierzchołki.

Definicja 55. (Graf acykliczny) *Graf, w którym nie występuje żadna ścieżka postaci $\langle v_0, v_1, \ldots, v_n, v_0\rangle$ nazywamy grafem acyklicznym.*

Przykład. W grafie z rysunku 3.2 można bez trudu wskazać ścieżkę, która rozpoczyna i kończy się tym samym wierzchołkiem np. $\langle sp, s1, s2, sp\rangle$. Graf ten nie jest zatem acykliczny. Grafem acyklicznym jest natomiast graf z rysunku 3.1(a).

Definicja 56. (Łańcuch) *Niech $G = (V, E)$ będzie grafem. Jeżeli $V = \{v_0, v_1, \ldots, v_n\}$ i $E = \{(v_{i-1}, v_i) : 1 \leqslant i \leqslant n\}$, to graf taki nazywamy łańcuchem.*

Przykład. Na rysunku 3.3 przedstawiony został graf, którego wierzchołkami są wyrazy zdania „Ich ziehe das szwarze Stromkabel ab". Wierzchołki odpowiadające sąsiadującym ze sobą wyrazom zdania zostały połączone krawędziami. Graf ten jest łańcuchem.

Definicja 57. (Drzewo ukorzenione) *Spójny graf acykliczny, w którym wyróżniono jeden wierzchołek, zwany* korzeniem drzewa*, nazywamy drzewem ukorzenionym.*

Wierzchołek v, który sąsiaduje w drzewie tylko z jednym wierzchołkiem, nazywamy liściem.

Przykład. Spójna składowa grafu z rysunku 3.1(a) zawierająca wierzchołek S jest drzewem ukorzenionym, o korzeniu w S.

Definicja 58. (Cięcie krawędziowe) *Niech G będzie grafem spójnym. Zbiór krawędzi, których usunięcie z grafu G powoduje, że przestaje być on spójny, nazywamy* cięciem krawędziowym.

Przykład. W spójnym grafie acyklicznym każda krawędź jest cięciem krawędziowym.

3.3. Reprezentacja danych

Algorytm uczący przetwarza korpus dwujęzyczny wyrównany z dokładnością do wyrazu, w którym zdania docelowe zostały oznaczone składniowo. Reprezentacją korpusu jest ciąg grafów wyrównania.

Oznaczenia:

$\langle w_1, w_2, \ldots, w_n \rangle$	zdanie — ciąg tokenów (wyrazów)
S	zdanie źródłowe (w języku, z którego odbywać się będzie tłumaczenie)
T	zdanie docelowe (w języku, na który odbywać się będzie tłumaczenie)
$V(G)$	zbiór wierzchołków grafu G
$E(G)$	zbiór krawędzi grafu G

Definicja 59. (Graf zdania) *Grafem SG_W zdania $W = \langle w_1, w_2, \ldots, w_n \rangle$ nazywamy łańcuch, taki że:*

$$V(SG) = \{w_1, w_2, \ldots, w_n\}$$
$$E(SG) = \{(w_1, w_2), (w_2, w_3), \ldots, (w_{n-1}, w_n)\}$$

Przykład. Graf zdania ⟨`ich`,`ziehe`,`das`,`szwarze`,`Stromkabel`,`ab`⟩ pokazany jest na rysunku 3.3.

Definicja 60. (Krawędzie wyrównania) *Niech SG_S i SG_T będą odpowiednio grafami zdania źródłowego S i docelowego T.*
Zbiorem krawędzi wyrównania $A_{T,S}$ zdania docelowego T i źródłowego S nazywamy zbiór uporządkowanych par (t, s), gdzie $t \in V(SG_T)$, a $s \in V(SG_S)$.

Przykład. Dla pary zdań:

$T =$ ⟨`Członek`,`Komisji`⟩

$S =$ ⟨`Membre`,`de`,`la`,`Commission`⟩

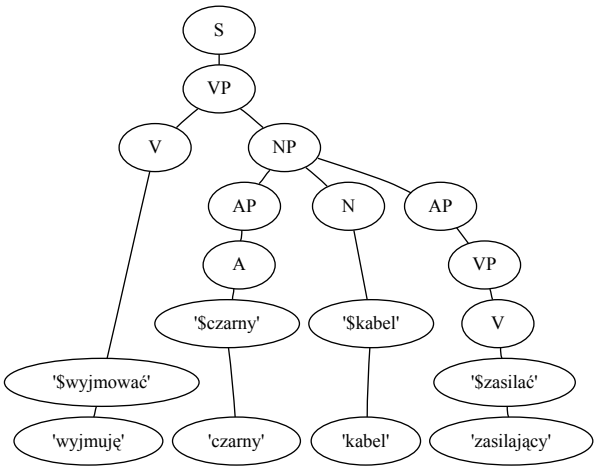
zbiór krawędzi wyrównania $A_{(T,S)}$ może być określony następująco:

$\big\{$(`Członek`,`Membre`), (`Komisji`,`de`), (`Komisji`,`la`), (`Komisji`,`Commission`)$\big\}$

Definicja 61. (Drzewo parsowania) *Drzewem parsowania PT_W zdania W nazywamy ukorzenione drzewo takie, że $V(SG_W)$ są wszystkimi liśćmi tego drzewa.*

Wierzchołki wewnętrzne drzewa parsowania nazywamy wierzchołkami parsowania.

Przykład. Drzewo parsowania zdania ⟨`wyjmuję`,`czarny`,`kabel`,`zasilający`⟩ pokazane jest na rysunku 3.4.



Rysunek 3.4. Przykładowe drzewo parsowania.

Definicja 62. (Graf wyrównania) *Grafem wyrównania $AG_{T,S}$ zdania docelowego T i źródłowego S nazywamy skierowany graf taki, że*

$$AG_{T,S} = PT_T + A_{T,S} + SG_S$$

Dla większej przejrzystości na rysunkach przedstawiających grafy wyrównania nie będą zaznaczane krawędzie grafu zdania źródłowego.

Przykład. Graf wyrównania dla zdań:

$T = \langle$`wyjmuję`,`czarny`,`kabel`,`zasilający`$\rangle$

$S = \langle$`ich`,`ziehe`,`das`,`szwarze`,`Stromkabel`,`ab`$\rangle$

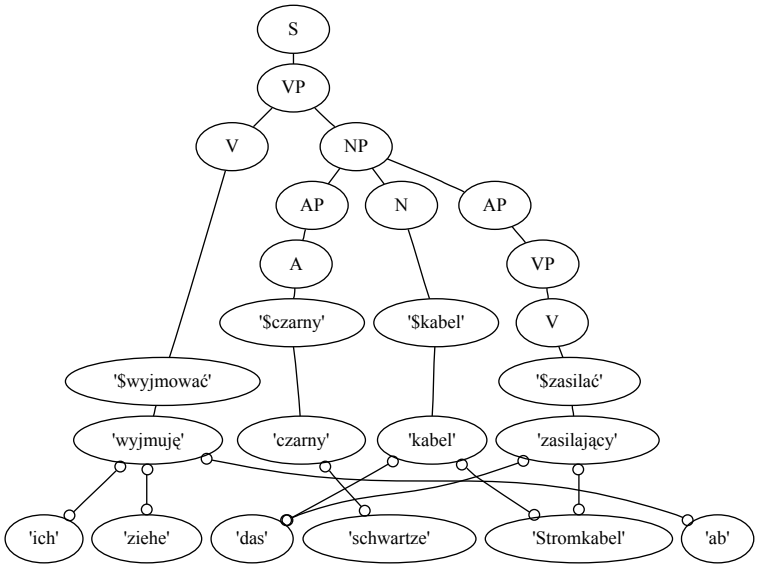
pokazany jest na rysunku 3.5.

3.4. Algorytm uczący

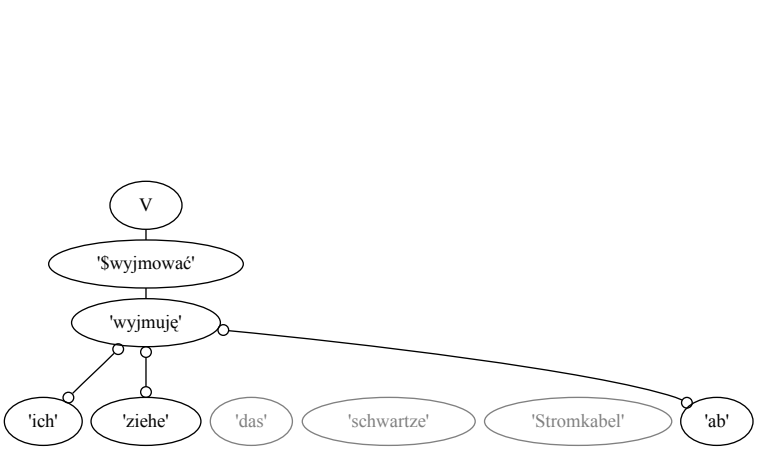
Przedstawiony poniżej algorytm uczący dokonuje rozkładu grafu wyrównania na składowe, zwane grafami indukującymi regułę. Grafy te zostają zindeksowane w bazie danych, z której korzysta algorytm tłumaczący.

Nieformalnie można powiedzieć, że graf indukujący regułę, to taki spójny podgraf grafu wyrównania, który „przetwarza" ciągły fragment zdania źródłowego.

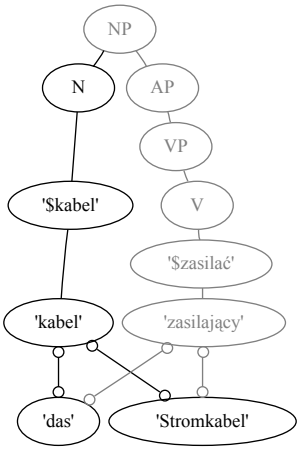
Przykład. Na rysunku 3.6 pokazano pewne podgrafy grafu wyrównania z rysunku 3.5, które nie są grafami indukującym regułę. Aby zilustrować, dlaczego te podgrafy nie są grafami indukującymi regułę, na rysunku zaznaczono kolorem szarym „okoliczne" węzły i krawędzie grafu wyrównania.



Rysunek 3.5. Przykładowy graf wyrównania.

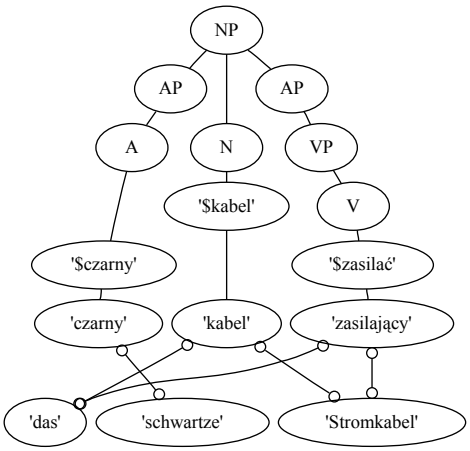


(a) Pokryty jest nieciągły fragment zdania źródłowego.



(b) „Gubione" są pewne krawędzie wyrównania.

Rysunek 3.6. Przykładowe grafy, które nie są grafami indukującym regułę.



Rysunek 3.7. Przykładowy graf indukujący regułę.

Podgraf na rysunku 3.6(a) jest spójny, ale nie rozciąga się ponad ciągłym fragmentem zdania źródłowego (pomija wierzchołki: `das`, `schwarze` i `Stromkabel`).

Podgraf na rysunku 3.6(b) jest co prawda spójny i rozciąga się ponad podciągiem zdania źródłowego, ale nie opisuje on w pełni tłumaczenia tego fragmentu, gdyż „gubi" pewne krawędzie wyrównania („zgubionymi" krawędziami są (`zasilający`,`das`) i (`zasilający`,`Stromkabel`)).

Definicja 63. (Graf indukujący regułę tłumaczenia) *Niech $AG_{T,S}$ będzie grafem wyrównania zdania źródłowego S i docelowego T. Niech $v \in PT_T$ będzie poprzednikiem (ojcem) wierzchołka wewnętrznego p w drzewie parsowania.*

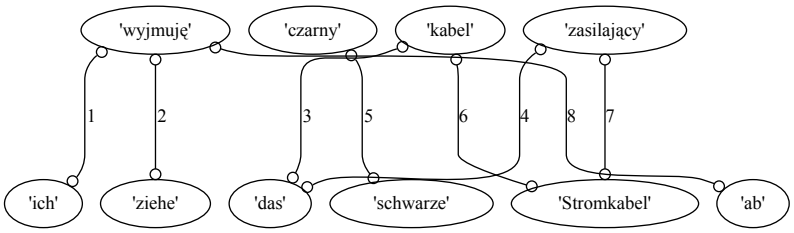
Grafem indukującym regułę $RIG_{p,AG_{T,S}}$ nazywamy spójną składową grafu $AG - \{(v,p)\}$ zawierającą p, dla cięcia krawędziowego złożonego z jednej lub dwóch krawędzi grafu zdania źródłowego SG_S.

Przykład. Jeden z podgrafów grafu z rysunku 3.5, który spełnia definicję 63 pokazany jest na rysunku 3.7.

W myśl przyjętych w powyższej definicji oznaczeń p = NP, v = VP, a cięciem krawędziowym grafu $AG - \{(\mathrm{VP}, \mathrm{NP})\}$ są krawędzie:

$$(\text{ziehe,das}), (\text{Stromkabel}, \text{ab}) \in SG_S$$

Algorytm uczący znajduje podgrafy indukujące reguły w zadanym grafie wyrównania obliczając dla każdego wierzchołka grafu wartości wektorów binarnych *span* i *mask*, zdefiniowanych w rozdziale 3.4.1. W dalszej części pracy wskazane zostaną warunki, jakie muszą spełniać te wektory, aby graf zakorzeniony w danym węźle był grafem indukującym regułę.



Rysunek 3.8. Numeracja przykładowych krawędzi wyrównania zgodnie z relacją 3.1.

3.4.1. Wektory *span* i *mask*

Aby móc zdefiniować wektory, na których będzie operował algorytm uczący, konieczne jest ponumerowanie krawędzi wyrównania w porządku „od lewej do prawej" od 1 do $|A|$. Kolejne numery otrzymują krawędzie łączące „wysunięty najbardziej na lewo" token zdania źródłowego z położonym „najbardziej na lewo" tokenem zdania docelowego.
Mówiąc precyzyjnie, krawędź wyrównania $e = (t, s) \in A$ oznaczona zostaje numerem $n(e) \in [1, |A|]$ tak, aby dla $j \neq l$ zachodziło:

$$n\left((t_i, s_j)\right) < n\left((t_k, s_l)\right) \Leftrightarrow \begin{cases} \langle t_i, \ldots, t_k \rangle, \text{ gdy } j = l \\ \langle s_j, \ldots, s_l \rangle, \text{ gdy } j \neq l \end{cases} \tag{3.1}$$

gdzie $(t_i, s_j), (t_k, s_l) \in A$, a $\langle t_i, \ldots, t_k \rangle$ i $\langle s_j, \ldots, s_l \rangle$ są ścieżkami odpowiednio w SG_T i SG_S.

Przykład. Numeracja krawędzi wyrównania grafu z rysunku 3.5 zgodna z relacją 3.1 została pokazana na rysunku 3.8.

Dla każdego wierzchołka v w grafie wyrównania algorytm określa wartości dwóch wektorów binarnych $span(v)$ i $mask(v)$ długości równej ilości krawędzi wyrównania ($|span(v)| = |mask(v)| = |A|$).
Dla każdego wierzchołka źródłowego $s \in V(SG_S)$ wektory zdefiniowane są następująco:

$$span(s) = 0 \tag{3.2}$$

$$mask(s) = [m_{|A|}, .., m_1] : m_i = \begin{cases} 1, \exists_{e=(s,t)\in A}\, i = n(e) \\ 0, \text{w przeciwnym razie} \end{cases} \tag{3.3}$$

Dla każdego wierzchołka docelowego $t \in V(SG_T)$ wektory zdefiniowane są następująco:

$$span(t) = [m_{|A|}, .., m_1] : m_i = \begin{cases} 1, \exists_{e=(s,t)\in A}\, i = n(e) \\ 0, \text{w przeciwnym razie} \end{cases} \tag{3.4}$$

$$mask(t) = \bigvee_{(t,s)\in A_{T,S}} mask(s) \tag{3.5}$$

Dla pozostałych wierzchołków grafu wyrównania, a więc dla każdego wierzchołka parsowania $p \in V(PT_T) - V(SG_T)$, wektory obliczane są następująco:

$$span(p) = \bigvee_{(p,v)\in E(PT_T)} span(n) \tag{3.6}$$

$$mask(p) = \bigvee_{(p,v)\in E(PT_T)} mask(n) \tag{3.7}$$

3.4.2. Własności wektorów *span* i *mask*

Twierdzenie 1. *Dla każdego wierzchołka parsowania $p \in V(PT_T)$, $mask(p) = span(p)$ wtedy i tylko wtedy, gdy krawędź $e = (v, p) \in E(PT_T)$, gdzie v jest ojcem p, jest krawędzią cięcia grafu $PT_T + A_{T,S}$.*

Dowód. Niech R będzie podgrafem grafu $PT_T + A_{T,S}$ indukowanym wierzchołkowo przez p i wierzchołki z niego osiągalne.

($\Rightarrow$) Przypuśćmy, że $mask(p) = span(p)$ i że usunięcie krawędzi e nie powoduje podziału grafu $PT_T + A_{T,S}$ na rozłączne składowe.
Ponieważ PT_T jest drzewem, a w drzewie każda krawędź jest krawędzią cięcia, więc musi istnieć krawędź wyrównania $e' = (t, s)$ taka, że $t \notin V(R) \wedge s \in V(R)$.

Wektor $span(p)$, obliczony zgodnie z równaniem 3.6, ostatecznie równy jest:

$$span(p) = \bigvee_{v\in V(R)\cap V(SG_T)} span(v) \tag{3.8}$$

Zgodnie z równaniem 3.4: $\forall_{v\in V(SG_T)\setminus\{t\}}\ span(v)_{n(e')} = 0$, więc $span(p)_{n(e')} = 0$.
Założono, że $mask(p) = span(p)$, więc $mask(p)_{n(e')} = 0$. Poprzez rozwinięcie rekurencji w równaniu 3.7 otrzymuje się:

$$mask(p) = \bigvee_{x\in V(R)\cap V(SG_S)} mask(x) \tag{3.9}$$

Zgodnie z równaniem 3.1: $mask(s)_{n(e')} = 1$, a skoro zgodnie z założeniem $s \in V(R)$, to $mask(p)_{n(e')} = 1$.

To oznacza sprzeczność, gdyż $span(p) = 0$ i $mask(p) = 1$, a założono, że wektory mają być równe.

($\Leftarrow$) Teraz załóżmy, że krawędź e jest krawędzią cięcia, ale $mask(p) \neq span(p)$.

Jeżeli $mask(p) \neq span(p)$ to $\exists i :\ mask(p)_i \neq span(p)_i$. Niech (t, s) będzie krawędzią o numerze i.
O krawędzi tej wiemy również, że $t, s \in V(R)$, gdyż $\{e\}$ jest cięciem krawędziowym, a wartości wektorów dla każdego wierzchołka parsowania wyliczane są zawsze z wierzchołków z niego osiągalnych (o czym mówią równania 3.8 i 3.9).

Załóżmy, że $span(p)_i = 0$ i $mask(p)_i = 1$. Z równania 3.8 otrzymujemy $1 = mask(p)_i = mask(s)_i$. Z definicji wektorów *span* i *mask* wiemy, że $mask(s)_i = span(t)_i$. Skoro $t, s \in V(R)$ więc zachodzi $1 = span(t)_i = span(p)_i$, co przeczy założeniu, że $span(p)_i = 0$.

Podobne rozumowanie w przypadku $span(p)_i = 1$ i $mask(p)_i = 0$ prowadzi również do sprzeczności.

Twierdzenie 2. *Niech R będzie podgrafem grafu $PT_T + A_{T,S}$ indukowanym wierzchołkowo przez p i wierzchołki z niego osiągalne, a $span(p) = mask(p)$.*
Wierzchołki źródłowe grafu R tworzą ścieżkę w SG_S, a wierzchołki docelowe — ścieżkę w SG_T wtedy i tylko wtedy, gdy

$$span(p) = mask(p) = [\underbrace{0\ldots0}_{a}, \underbrace{1\ldots1}_{b-a}, \underbrace{0\ldots0}_{|A_{T,S}|-b}]$$

gdzie $0 \leqslant a < b \leqslant |A_{T,S}|$.

Dowód. ($\Leftarrow$) Załóżmy, że wektory *span* i *mask* dla wierzchołka p mają postać `0*1+0*` (zero lub więcej zer, co najmniej jedna jedynka i zero lub więcej zer), ale wierzchołki źródłowe grafu R nie tworzą ścieżki w grafie zdania źródłowego.

Oznacza to, że w grafie zdania źródłowego musi istnieć łańcuch długości m taki, że pierwszy i ostatni węzeł tego łańcucha $s_1, s_m \in V(SG_S)$ są węzłami w grafie R, natomiast jeden lub więcej wewnętrznych węzłów tego łańcucha nie jest węzłem w grafie R. Na podstawie twierdzenia 1 (a w szczególności równania 3.9) wiemy, że wektor *mask* dla wierzchołka p grafu R będzie miał następujące wartości:

$$\begin{aligned} \forall (t, s_1) \in A_{T,S}: &\quad mask(p)_{n((t,s_1))} = 1 \\ \forall i \in (1, m)\ \forall (t, s_i) \in A_{T,S}: &\quad mask(p)_{n((t,s_i))} = 0 \\ \forall (t, s_1) \in A_{T,S}: &\quad mask(p)_{n((t,s_m))} = 1 \end{aligned}$$

Na podstawie relacji 3.1 wiemy więc, że wektor *mask* przyjmie postać `10+1` (jedynka, co najmniej jedno zero i jedynka), a to przeczy założeniu, że wektor *mask* ma być postaci `0*1+0*`.

($\Rightarrow$) Skoro wierzchołki $s_i \in V(R) \cap V(SG_S)$ tworzą łańcuch w grafie zdania źródłowego, a krawędzie numerowane są „od lewej do prawej" zgodnie z relacją 3.1, to krawędzie $(t_j, s_i) \in A_{T,S}$ otrzymały kolejne numery.
Niech $a = min\{n((t_j, s_i))\}$ i $b = max\{n((t_j, s_i))\}$.

Zgodnie z równaniem 3.9 otrzymujemy $\forall_{i \in [a,b]} [mask(p)]_i = 1$.
Dla każdego węzła źródłowego $s \notin V(R)$ możemy wskazać ścieżkę w grafie źródłowym $\langle s, \ldots, s_i \rangle$ lub $\langle s_i, \ldots, s \rangle$. Tak więc zgodnie z relacją 3.1 zachodzi $\forall_i \forall_{(t,s), s \neq s_i} n((t, s)) < a$ lub $\forall_i \forall_{(t,s), s \neq s_i} n((t, s)) > b$, a więc $\forall_{i<a} [mask(p)]_i = 0$ lub $\forall_{i>b} [mask(p)]_i = 0$.

Tak więc wektor *mask* ma postać taką, jak wskazano w twierdzeniu.

3.4.3. Aspekty implementacyjne

Algorytm uczący dokonuje początkowego określania wartości wektorów binarnych z pominięciem jawnego numerowania krawędzi wyrównania.

INITIALIZE-VECTORS($AG_{T,S}$)
1 $zero \leftarrow [\underbrace{0 \ldots 0}_{|A|}]$
2 $label \leftarrow [\underbrace{0 \ldots 0}_{|A|-1} 1]$
3 **for** $t \in V(SG_T)$
4 **do** $span[t] \leftarrow zero$
5 $mask[t] \leftarrow zero$
6 **for** $i \leftarrow 1$ **to** $len(S)$
7 **do** $span[s_i] \leftarrow zero$, gdzie $s_i \in V(SG_S)$
8 $mask[s_i] \leftarrow zero$
9 **for** $j \leftarrow 1$ **to** $len(T)$
10 **do if** $(s_i, t_j) \in A$, gdzie $t_j \in V(SG_T)$
11 **then** $mask[s_i] \leftarrow mask[s_i] \vee label$
12 $span[t_j] \leftarrow span[t_j] \vee label$
13 $mask[t_j] \leftarrow mask[t_j] \vee mask[s_i]$
14 $label \leftarrow$ L-SHIFT($label$)

Przez $\vee$ oznaczono operację binarnego „lub", natomiast funkcja L-SHIFT to operacja binarnego „przesunięcia w lewo" określona następująco:

L-SHIFT($[a_1, a_2, \ldots, a_n]$)
1 **return** $[a_2, a_3, \ldots, a_n, 0]$

Procedura INITIALIZE-VECTORS dla reprezentacji macierzowej grafu wyrównania ma złożoność $O(n * m)$. Wynika to wprost z warunków iteracji w liniach 6 i 9.
Przy zastosowaniu reprezentacji listowej takiej, że z każdym wierzchołkiem źródłowym związana jest lista wierzchołków docelowych sąsiadujących z tym wierzchołkiem, posortowana względem pozycji w zdaniu docelowym, procedurę można łatwo zmodyfikować tak, aby złożoność jej wynosiła $\theta(|A_{T,S}|)$. W tym celu wystarczy zastąpić warunek iteracji w linii 9, iteracją po liście związanej z wierzchołkiem s_i.

Iteracja w liniach 3–5 może być wykonywana równolegle. Poza tym wykonanie procedury nie może zostać zrównoleglone.

Twierdzenie 3. *Poprawność działania* INITIALIZE-VECTORS
Procedura INITIALIZE-VECTORS *w sposób iteracyjny oznacza wierzchołki zdania źródłowego i docelowego wektorami, które po jej wykonaniu mają postać określoną równaniami 3.2–3.5.*

Dowód. Procedura jest poprawna, gdyż:

1. wszystkie wektory *span* i *mask* mają długość $|A|$,
2. wektor *label* zawsze ma ustawiony tylko jeden bit,
3. odwiedzając każdą krawędź wektory *span* i *mask* modyfikowane są jedynie na jednej pozycji — pozycji ustawionego bitu wektora *label*,
4. każda krawędź wyrównania odwiedzana jest tylko raz i podczas każdych odwiedzin wektor *label* ma inną wartość,
5. krawędzie wyrównania odwiedzane są w kolejności zgodnej z porządkiem 3.1.

(1) Wartości wektorów wyliczane są z wektorów *zero* i *label*, które mają długość $|A|$, a każda z operacji zastosowanych w procedurze zachowuje długość argumentów. Stąd wektory *span* i *mask* mają długość równą $|A|$.

(2) Wektor *label* modyfikowany jest jedynie w linii 14 operacją L-SHIFT. Z definicji tej operacji widać, że jedyne ustawione bity w wartości zwracanej mogą pochodzić z argumentu operacji. Skoro więc wektor *label* inicjowany jest pojedynczym ustawionym bitem, więc nigdy nie będzie miał on więcej niż jeden ustawiony bit.

(3) Poza inicjalizacją (linie 4, 5, 7 i 8) wektory *span* i *mask* modyfikowane są tylko w liniach 11–13. Modyfikacja ta polega na wykonaniu binarnego „lub" na sobie i pewnym innym wektorze. Ponieważ prawdą jest, że $a \vee 0 = a$, więc wpływ na wynik operacji mają jedynie ustawione bity. W przypadku linii 11 i 12 będzie to jedyny bit wektora *label* (patrz: 2), a więc zmiana wartości wektorów *span* i *mask* będzie miała miejsce tylko na jednej pozycji.
W linii 13 źródłem zmiany wartości wektora $mask[t_j]$ jest wektor $span[s_i]$, ale ten mógł ulec zmianie w linii 11 jedynie na jednej pozycji.

(4) Linie 11–14 wykonywane są wtedy i tylko wtedy, gdy $(s_i, t_j) \in A$. Dla ustalonego wierzchołka $s_i \in V(uSG_S)$, każdy wierzchołek $t_j \in V(uSG_T)$ rozpatrywany jest tylko raz (gwarantuje to warunek iteracji w linii 9). Podobnie — każdy wierzchołek $s_i \in V(uSG_S)$, zgodnie z warunkiem iteracji w linii 6, rozpatrywany jest tylko raz. Tak więc każda kombinacja wierzchołków s_i i t_j rozpatrywana jest w warunku w linii 10 dokładnie raz, a więc każda krawędź wyrównania zostaje odwiedzona tylko jeden raz.

Wartość wektora *label* zmieniana jest jedynie w linii 14. Zachodzi

$$\forall a \neq 0 : a \neq \text{L-SHIFT}(a)$$

Ponieważ linia 14 wykonywana jest dokładnie $|A|$ razy, a wektor *label* ma wartość początkową $[\underbrace{0 \ldots 0}_{|A|-1}1]$, więc wartość 0 zostanie osiągnięta dopiero przy odwiedzeniu ostatniej krawędzi. Tak więc podczas odwiedzin każdej krawędzi wartość wektora *label* jest inna.

(5) Równania 3.3 i 3.4 wiążą numer krawędzi z pozycją w wektorach *mask* i *span*. Procedura nie wylicza jawnie numeru dla każdej z krawędzi, lecz od razu modyfikuje wektory binarne na pozycji odpowiadającej numerowi krawędzi.
Jeżeli procedura odwiedza właśnie krawędź (t_k, s_i), to zgodnie warunkami iteracji w liniach 6 i 9, odwiedzi ona następnie krawędź (t_x, s_i) (o ile taka istnieje) , gdzie x to indeks wierzchołka docelowego sąsiadującego z s_i takiego, że ścieżka $\langle t_k, \ldots, t_x \rangle$ jest najkrótsza. Jeżeli krawędź (t_x, s_i) nie istnieje, to procedura przejdzie do krawędzi (t_y, s_{i+1}) (o ile taka istnieje), gdzie y to indeks wierzchołka docelowego sąsiadującego z s_{i+1} takiego, że ścieżka $\langle t_1, \ldots, t_y \rangle$ jest najkrótsza.
Porządek 3.1 jest więc zachowany w każdym kroku procedury.

Wartości wektorów dla pozostałych wierzchołków grafu wyrównania oblicza procedura przechodząca graf w głąb. Podczas tego samego przejścia grafu wyrównania, oznaczone zostają wierzchołki, dla których spełnione są warunki, o których mowa w twierdzeniu 2.

```
COMPUTE-VECTORS(AG_{T,S})
 1  zero ← [0...0]  (|A| razy)
 2  ready[ korzeń drzewa PT_T] ← FALSE
 3  S ← ∅
 4  PUSH(S, korzeń drzewa PT_T)
 5  while S ≠ ∅
 6     do v ← POP(S)
 7        if ready[v] = FALSE
 8          then ready[v] ← TRUE
 9               PUSH(S, v)
10               for v' ∉ V(SG_T) : (v, v') ∈ E(PT_T)
11                  do ready[v'] ← FALSE
12                     PUSH(S, v')
13          else  if ∃v' : (v, v') ∈ E(PT_T)
```

```
14                then mask[v] ← span[v] ← zero
15                     for v' : (v, v') ∈ E(PT_T)
16                         do mask[v] ← mask[v] ∨ mask[v']
17                            span[v] ← span[v] ∨ span[v']
18             if mask[v] = span[v] = [ 0*1+0* ]
19                then RIG[v] ← TRUE
20                else RIG[v] ← FALSE
```

Użyte w procedurze Compute-Vectors procedury Push i Pop to standardowe operacje na stosie — umieszczenie i zdjęcie elementu ze szczytu stosu.

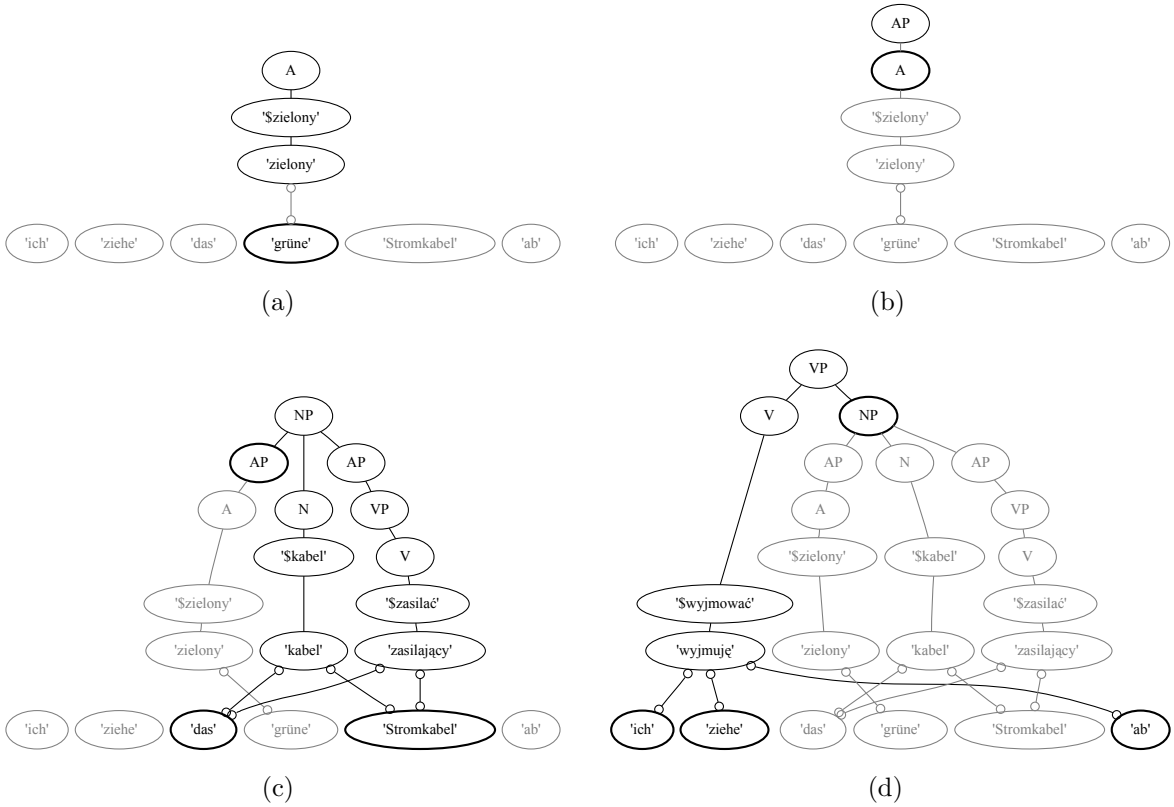
Główna pętla procedury (linie 5–20) wykonuje się tak długo, jak długo niepusty jest stos S. Przy każdej iteracji tej pętli ze stosu zdejmowany jest (linia 6) jeden wierzchołek (v). Jeżeli nie był on wcześniej odwiedzony (linia 7), zostaje on oznaczony jako odwiedzony (linia 8) i ponownie odłożony na stos (linia 9), a jego dzieci niebędące liśćmi w drzewie PT_T (linia 10), zostają oznaczone jako nieodwiedzone (linia 11) i umieszczone na stosie (linia 12). Gdy ze stosu zostanie zdjęty wierzchołek odwiedzony ($ready[v]$ = TRUE), wtedy obliczone zostają wartości wektorów *span* i *mask* (linie 13–16). Jeżeli wektory okażą się równe i będą miały postać, o której mowa w twierdzeniu 2, to wierzchołek zostaje oznaczony jako korzeń grafu indukującego regułę (linia 18).

Procedura Compute-Vectors ma złożoność rzędu $O(V(PT_T))$, gdyż każdy wierzchołek drzewa PT_T umieszczany jest maksymalnie dwukrotnie na stosie S, a za każdym razem procedura wymaga dostępu jedynie do węzłów bezpośrednio sąsiadujących z aktualnie analizowanym wierzchołkiem.

Procedura Compute-Vectors może zostać łatwo zmodyfikowana tak, aby możliwe było zrównoleglenie jej wykonania. W bieżącej postaci procedura rozpoczyna wykonanie do korzenia drzewa parsowania (linie 2–4). Gdyby pierwszy wierzchołek odkładany na stosie (linia 4) zadany był parametrem wywołania procedury, wtedy w iteracji w linii 10 mogłoby się odbywać równoległe rekurencyjne wywołanie procedury dla każdego z dzieci bieżącego węzła. Po zakończeniu obliczeń we wszystkich równolegle wykonywanych rekurencyjnych wywołaniach, działanie procedury głównej mogłoby zostać wznowione.

3.5. Algorytm tłumaczący

Działanie algorytmu tłumaczącego polega na rekonstrukcji grafu wyrównania dla danego zdania źródłowego na podstawie bazy grafów indukujących reguły, wyznaczonych przez algorytm uczący.



Rysunek 3.9. Kolejne etapy konstruowania tłumaczenia przykładowego zdania.

Przykład. Załóżmy, że wejściem do algorytmu tłumaczącego jest zdanie „Ich ziehe das grüne Stromkabel ab".

Dysponując bazą grafów indukujących regułę pozyskanych między innymi z grafu wyrównania przedstawionego na rysunku 3.5, algorytm tłumaczący jest w stanie skonstruować tłumaczenie zdania wejściowego, dopasowując grafy z bazy do zdania wejściowego. Kolejne etapy tego procesu przedstawia rysunek 3.9.

Na rysunku 3.9 kolorem szarym zaznaczono węzły, które powstałe w jednym z poprzednich kroków algorytmu, a kolorem czarnym — nowo dodane węzły. Dodatkowo pogrubiono węzły, które obecne były w poprzednim kroku algorytmu oraz w (dopasowywanym w bieżącym kroku) grafie indukującym regułę; węzły te ulegają scaleniu.

Główną procedurą algorytmu tłumaczącego jest procedura Compute-Matrix. Jej działanie polega na systematycznym konstruowaniu grafu wyrównania poprzez scalanie rozłącznych grafów przy użyciu grafów indukujących reguły. Wejściem do procedury jest tablica M rozmiaru $n \times n$. Warunkiem wstępnym jest inicjacja tablicy poprzez wypełnienie wiersza o indeksie 0 referencjami do węzłów zdania źródłowego.

```
COMPUTE-MATRIX(M_{n×n})
 1  for i ← 0 to n − 1
 2      do M[1][i] ← EXPAND-CELL(M[0][i])
 3  for s ← 2 to n
 4      do for b ← 0 to n − s
 5             do Rs ← ∅
 6                for f ← 1 to s − 1
 7                    do R ← JOIN-CELLS(M[f][b], M[s − f][b + f])
 8                       PUSH(Rs, R)
 9                M[s][b] ← MERGE-RESULTS(Rs)
10                M[s][b] ← EXPAND-CELL(M[s][b])
```

Ponieważ definicja 63 mówi, że graf indukujący regułę ma „pokrywać" podciąg zdania źródłowego, komórka tablicy $M[i][j]$ przechowuje referencje do wszystkich grafów wyrównania, jakie algorytm zdołał zbudować nad podciągiem zdania źródłowego długości i zaczynającym się od tokenu j. W związku z tym w trakcie działania procedury wypełniona zostaje w najgorszym przypadku jedynie połowa tablicy (do przekątnej włącznie). Następuje to wskutek wykonania iteracji w liniach 3 i 4.

Grafy wyrównania dla podciągu o długości i zaczynające się od tokenu j znajdowane są w wyniku scalania grafów dla podciągu długości k zaczynającego się od tokenu j z grafami dla podciągu długości $i - k$ zaczynającego się od tokenu $j + k$. Za przeanalizowanie wszystkich możliwych par odpowiada iteracja w linii 6.

Może się zdarzyć, że ten sam rezultat można uzyskać na więcej niż jeden sposobów. Dlatego wyniki scalania dwóch grafów (linia 7) analizowane są w celu znalezienia powtórzeń w linii 9.

Oprócz scalania dwóch grafów, dopuszcza się również operację rozbudowywania pojedynczego grafu za pomocą grafu indukującego regułę. W liniach 1–2 rozbudowywane są węzły źródłowe, a w linii 10 — grafy uzyskane w wyniku scalania.

Na podstawie iteracji w liniach 3, 4 i 6 złożoność procedury COMPUTE-MATRIX można oszacować na $O(n^3)$.

Wykonanie procedury można zrównoleglić. Należy zauważyć, że dla konstruowania grafów wyrównania nad podciągiem zdania źródłowego o długości i wymagane są grafy wyrównania zbudowane nad podciągiem o długości $i - 1$ i mniejsze. W związku z tym dla ustalonej wartości zmiennej s obliczenia w liniach 5–10 mogą zostać wykonane równolegle dla każdej wartości b.

Procedura Expand-Cell analizuje wszystkie grafy wyrównania zbudowane nad podciągiem o długości i zaczynającym się od tokenu j zdania źródłowego. Celem procedury jest powiększenie grafu o dodatkowe wierzchołki i krawędzie, bez zwiększania długości ciągu, nad którym graf jest zbudowany.

```
EXPAND-CELL(C)
 1  S ← ∅
 2  for ∀c ∈ C
 3     do if expanded(c) = FALSE
 4           then PUSH(S, c)
 5  while S ≠ ∅
 6     do s ← POP(S)
 7        expanded(s) ← TRUE
 8        N ← FIND-MATCH(s)
 9        if N ≠ ∅
10           then for ∀n ∈ N
11                   do m ← s + n
12                      if ∃m′ ∈ C : root(m) = root(m′)
13                         then PUSH(alternatives[m′], m)
14                         else joinable(m) ← joinable(n)
15                              appendable(m) ← appendable(n)
16                              partial(m) ← FALSE
17                              C ← C ∪ m
18                              if expanded(n) ≠ TRUE
19                                 then PUSH(S, m)
```

Procedura Expand-Cell używa stosu S, na który odkładane są grafy, które nie zostały jeszcze rozbudowane (linie 4 i 19). To, czy dany graf został już rozbudowany, określa atrybut *expanded*. Sprawdzenie wartości tego atrybutu przed dodaniem grafów z komórki C do stosu S jest konieczne ze względu na to, że w innym miejscu (np. w procedurze Join-Cells) może zostać stwierdzone, że dany graf nie może zostać rozbudowany.

W liniach 14–16 algorytm przepisuje wartości atrybutów grafu indukującego regułę n do nowozbudowanego grafu m. Atrybuty te pochodzą z bazy grafów indukujących regułę. Są one istotne dla procedury Join-Cells i tam zostały opisane.

Wyszukanie pasujących do danego grafu wyrównania s grafów indukujących regułę realizowane jest przez procedurę Find-Match. W linii 11 z każdego dopasowania n i bieżącego grafu wyrównania s tworzony jest nowy (większy) graf wyrównania m.

Jest możliwe, że nowozbudowany graf m jest tożsamy z pewnym innym grafem, który został już wcześniej skonstruowany dla tego samego podciągu. Ewentualność ta sprawdzana jest w warunku w linii 12.

Procedura JOIN-CELLS realizuje scalanie dwóch grafów przy użyciu grafów indukujących regułę.

```
JOIN-CELLS(C_1, C_2)
 1  J ← ∅
 2  for ∀c ∈ C_1
 3      do if joinable(c) = FALSE
 4            then PUSH(J, c)
 5  A ← ∅
 6  for ∀c ∈ C_2
 7      do if appendable(c) = FALSE
 8            then PUSH(A, c)
 9  R ← ∅
10  for ∀j ∈ J
11      do for ∀a ∈ A
12           do N ← FIND-PARTIAL-MATCH(j, a)
13              for ∀n ∈ N
14                 do m ← j + a
15                    if partial(j) ≠ TRUE
16                      then m ← m + n
17                    expanded(m) ← TRUE
18                    joinable(m) ← TRUE
19                    appendable(m) ← FALSE
20                    partial(m) ← TRUE
21                    R ← R ∪ m
22              N ← FIND-PERFECT-MATCH(j, a)
23              for ∀n ∈ N
24                 do m ← j + a + n
25                    expanded(m) ← expanded(n)
26                    joinable(m) ← joinable(n)
27                    appendable(m) ← appendable(n)
28                    partial(m) ← FALSE
29                    R ← R ∪ m
30  return R
```

Procedura Join-Cells, w odróżnieniu od Expand-Cell, buduje grafy wyrównania z grafu indukującego regułę łącząc go dwoma lub więcej istniejącymi grafami.

Z komórek C_1 i C_2, a więc spośród grafów wyrównania ponad odpowiednimi (następującymi po sobie) podciągami zdania źródłowego, wybierane są te grafy, o których wiadomo, że mogą przyłączyć (atrybut *joinable*) inny graf (linie 2–4), oraz takie, które mogą zostać przyłączone (atrybut *appendable*) przez inny graf (linie 6–8). Wartości tych atrybutów pochodzą z bazy grafów indukujących regułę, gdzie zostały określone podczas jej tworzenia na podstawie analizy całego zbioru grafów indukujących regułę.

Każdą parę grafów taką, że pierwszy jest typu *joinable*, a drugi typu *appendable* (iteracje w liniach 10 i 11), algorytm stara się połączyć przy użyciu grafu indukującego regułę n. Graf n może zostać znaleziony na dwa sposoby — jako tzw. *częściowe i pełne dopasowanie*.

O *pełnym dopasowaniu* mówimy, gdy w wyniku połączenia grafu indukującego regułę z parą grafów wyrównania otrzymuje się graf wyrównania. Pełne dopasowania wyszukiwane są w linii 22. Graf będący wynikiem scalenia (linia 24) przejmuje w tym przypadku atrybuty po grafie indukującym regułę (linie 25–28).

O *częściowym dopasowaniu* mówimy, gdy możliwe jest połączenie grafu indukującego regułę z parą grafów wyrównania, ale wynikiem tej operacji nie jest graf wyrównania. Dzieje się tak w przypadku dużych grafów indukujących regułę, które są w stanie połączyć trzy lub więcej mniejsze grafy wyrównania. W takim przypadku algorytm scala ze sobą jedynie parę grafów wyrównania, dla której znalazł częściowe dopasowanie (linia 14). Wynik scalenia określony zostaje jako nieprzydatny dla procedury Expand-Cell (linia 17), jako przeznaczony do przyłączania dalszych grafów wyrównania (linia 18), ale sam niezdolny do bycia przyłączonym przez inny graf (linia 19).

Należy zauważyć, że grafy będące wynikiem częściowego dopasowania, podczas kolejnych wywołań procedury Join-Cells przez procedurę Compute-Matrix, mogą zostać powiększone o kolejne grafy wyrównania wskutek częściowego dopasowania. Można oczekiwać, że w podczas pewnego wywołania procedury Join-Cells dla grafów będących wynikami częściowego dopasowania zostaną znalezione zostanie pełne dopasowanie.

Rozdział 4

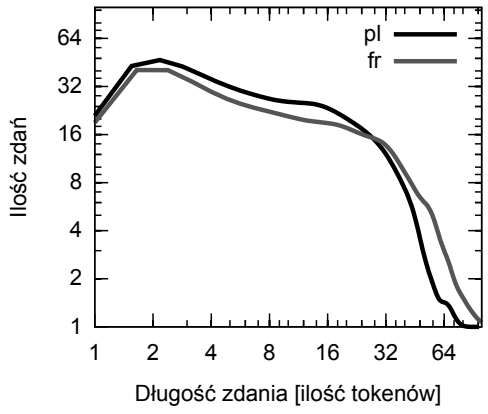
Weryfikacja poprawności działania opracowanych algorytmów

4.1. Przygotowanie danych treningowo–testowych

Algorytmy opisane w rozdziale 3 zostały poddane testom z użyciem niewielkiego korpusu polsko–francuskiego będącego podzbiorem korpusu *DGT–TM*.

Korpus DGT–TM to zbiór małych segmentów tekstu (całych zdań lub ich fragmentów) oraz ich tłumaczeń w 22 językach Unii Europejskiej. Został opublikowany przez Dyrekcję Generalną Tłumaczenia Komisji Europejskiej [1]. Korpus ten jest podzbiorem zasobów systemu *Euramis* (European advanced multilingual information system), który zawiera m. in. większość tekstów legislacyjnych Unii Europejskiej oraz orzeczeń Europejskiego Trybunału Sprawiedliwości.

Na potrzeby opisanych w tym rozdziale eksperymentów, z korpusu DGT–TM wybrano losowo 500 polsko–francuskich par zdań. Rozkłady długości zdań w zbiorze treningowo-testowym został przedstawione na rysunku 4.1. Część francuska zbioru treningowo–testowego zawierała 2483 różnych wyrazów, a polska — 3032.

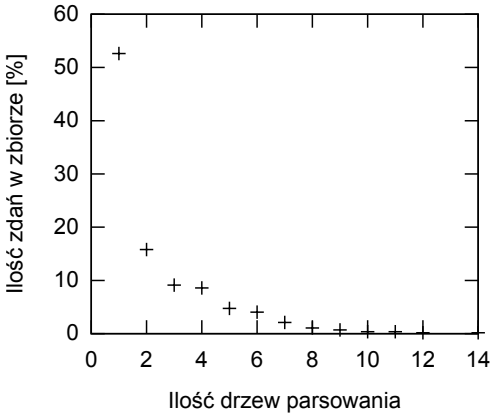


Rysunek 4.1. Rozkład długości zdań w zbiorze treningowo–testowym

Tworząc system tłumaczenia automatycznego należy mieć na uwadze „profil" tekstów do tłumaczenia, w których będzie on używany. Choć można wyobrazić sobie takie zastosowanie, w którym tłumaczone będą komunikaty o mniej więcej tej samej długości, to jednak w ogólnym przypadku należy liczyć się z tym, że system będzie musiał radzić sobie zarówno z bardzo krótkimi, jak i z wielokrotnie złożonymi zdaniami.

Przyjęcie założenia, że korpus treningowo–testowy ma mieć zbliżoną charakterystykę do całego korpusu DGT–TM spowodowało, że wśród wybranych zdań zalazły się zarówno nagłówki rozdziałów, jak i całe akapity dokumentów. Np. zdanie numer 29 w korpusie treningowo–testowym ma długość zaledwie 2, podczas gdy zdanie 296 numer ma długość 88 po stronie polskiej i 105 po stronie francuskiej.

	PL	*FR*
29	ZAŁĄCZNIK XI	ANNEXE VI
⋮		
296	Jednakże substancje wprowadzane do obrotu na etapie badań i rozwoju , z ograniczoną liczbą upoważnionych odbiorców , w ilościach ograniczonych do celów badań i rozwoju , ale które przekraczają 1 tonę rocznie na jednego producenta , kwalifikują się do wyłączenia na okres jednego roku , pod warunkiem że producent poda do wiadomości właściwym władzom każdego Państwa Członkowskiego ich nazwę , sposób etykietowania , ilość , miejsce ich produkcji oraz badań i rozwoju a także spełnia warunki wymagane przez te władze w stosunku do takich badań i rozwoju ;	Toutefois , les substances au stade de la recherche et développement mises sur le marché en des quantités limitées au but poursuivi par la recherche-développement mais supérieures à une tonne par an et par fabricant , auprès de clients enregistrés et en nombre limité , bénéficient d' une dérogation valable pour un an , à condition que le fabricant déclare leur identité , les données utilisées pour l' étiquetage et leur quantité aux autorités compétentes de chacun des États membres où ont lieu la fabrication , la recherche ou le développement et se conforme aux dispositions éventuelles imposées par ces autorités à cette recherche-développement ;



Rysunek 4.2. Procentowy udział w zbiorze treningowo–testowym zdań, które parsują się do jednego lub więcej drzew.

Podjęto próbę automatycznego urównoleglenia zbioru treningowo–testowego za pomocą narzędzi z pakietu GIZA++ [21]. Ponieważ te narzędzia powstały z myślą o dużych korpusach ich skuteczność na małych zbiorach zależy w dużej mierze od konkretnych danych. Z tego powodu na potrzeby badania zachowania algorytmów opisanych w rozdziale 3 dla każdej z par zdań w korpusie treningowo-testowym ręcznie określono odpowiedniość pomiędzy wyrazami polskimi i francuskimi.

Część polską zbioru treningowo–testowego poddano analizie składniowej przy użyciu regułowego parsera języka polskiego z systemu tłumaczącego Poleng. W przypadku 52% zdań polskich analizator składniowy zdołał przetworzyć zadany tekst jako całość. W pozostałych przypadkach wynikiem parsowania był las dwóch lub więcej drzew, co przedstawia wykres z rysunku 4.2. Sytuację, w której w wynikiem parsowania nie jest jedno drzewo można interpretować jako:

- błąd parsera, a więc przypadek kiedy gramatyka parsera nie jest dostatecznie bogata, aby zanalizować poprawne zdanie, lub przypadek kiedy w trakcie analizy składniowej osiągnięte zostały implementacyjne ograniczenia parsera (np. zajętość pamięci, czas działania, itp.); błąd parsera może pojawić się również w sytuacji, gdy w zdaniu wejściowym pojawią się poprawne, choć nieznane parserowi wyrazy np. obcojęzyczne wtrącenia;
- błąd wejścia, a więc przypadek kiedy zdanie poddane analizie składniowej nie jest poprawnym zdaniem języka naturalnego (w tym przypadku — polskiego).

Algorytmy z rozdziału 3 zaprojektowane zostały dla przypadku, gdy znana jest struktura składniowa całego zdanie docelowego. Z formalnego punktu widzenia znaczna część (48%) zbioru treningowo–testowego (ta, dla której wynikiem parsowania jest las) powinna zostać wykluczona z eksperymentu. To jednak spowodowałoby zmniejszenie korpusu do rozmiarów mniejszych niż założono przy projektowaniu algorytmów z rozdziału 3.

Aby spełnić założenie co do rozmiaru korpusu należałoby skompensować „sprawność" parsera większym zbiorem danych, co w zastosowaniach praktycznych może nie być możliwe. Z tego powodu na potrzeby eksperymentu drzewa tworzące las parsowania dla jednego zdania zostały połączone w jedno drzewo. Dokonano tego poprzez dodanie jednego wierzchołka jako korzenia nowego drzewa i połączenia go z korzeniami drzew tego lasu. Z jednej strony taka operacja wydaje się zasadna lingwistycznie (analizie został poddany pewien spójny fragment, który powinien zostać zanalizowany jako całość), jednak z drugiej strony pociąga za sobą pewne ryzyko. Należy być świadomym, że mało efektywne lub błędne działanie algorytmów z rozdziału 3 może być efektem włączenia do zbioru treningowego znacznie zakłóconych danych, a takimi bez wątpienia są zdania, których parser nie zdołał zanalizować w całości. W przypadku, gdy powstanie lasu spowodowane było błędem wejścia, w czasie trenowania algorytmów opisanych w rozdziale 3, mogą powstać reguły, które (o ile zostaną użyte) na etapie tłumaczenia mogą powodować generowanie niepoprawnych zdań polskich.

4.2. Implementacja testowa

Algorytmy opisane w rozdziale 3 zostały zaimplementowane w języku Perl[1], ze względu na jego wszechstronność (wieloparadygmatowość, dynamiczne typowanie) i kompatybilność z analizatorem składniowym systemu Poleng.

System Poleng udostępnia wynik analizy składniowej w postaci tekstowej, która zgodna jest formalnie ze składnią języka Perl.

Przykład. Poniżej przedstawiony został fragment drzewa parsowania frazy „Komora próżniowa" w formacie udostępnianym przez system Poleng:

```
[{  q[nr_in_parent] => -1, q[last_subtree] => 1, q[category] => q|FR|, q[label] => q||,
, , q^Score^ => 5000004, q^L^ => 1, q^Składnia^ => q^nil^, q^Beg^ => 0, q^O^ => 3,
q^R^ => q^ż^, q^L1^ => 1, q^S^ => q^room, artifact, organ^, q^Amb^ => 1, q^Przyim^ =>
q^brak^, q^P^ => q^mian^, q^R1^ => q^ż^, q^Length^ => 2, q^Sem1^ => q^device, body_part^,
q^Rp^ => q^ż^, q^Core^ => q^komora^ , q|subtrees| => [ {  q[nr_in_parent] => 0,
q[last_subtree] => 2, q[variant] => 1, q[category] => q|R|, q[label] => q|#|,  , , ,
q^Score^ => 2000002, q^L^ => 1, q^Składnia^ => q^nil^, q^Beg^ => 0, q^O^ => 3, ...
```

1. Perl to interpretowany język programowania wysokiego poziomu, do zastosowań ogólnych. Dystrybuowany jest na licencji *GNU General Public Licence*. Oficjalna strona internetowa tego projektu to `http://perl.org`

Nieformalnie składnię formatu wyjściowego systemu POLENG można przedstawić następująco. Zapis $[e_1, e_2, \ldots]$ oznacza listę elementów e_i, a $\{k_1 => v_1, k_2 => v_2, \ldots\}$ oznacza tablicę haszująca o kluczach k_i i odpowiadających im wartościach v_i.

Struktury mogą być zagnieżdżane, a więc lista może składać się nie tylko z typów prostych (tzw. *skalarów*, a więc liczb, napisów, itp.) ale również z innych list lub tablic haszujących. Podobnie wartościami w tablicy haszującej mogą być inne tablice haszujące lub listy. Zapisy $q[napis]$ oraz $q\hat{}napis\hat{}$ oznaczają stałą tekstową „napis".

Każdy wierzchołek drzewa parsowania reprezentowany jest w postaci tablicy haszującej. Spośród kluczy takiej tablicy wyróżnia się klucz „subtrees", którego wartością jest lista wierzchołków dzieci bieżącego wierzchołka. Pozostałe klucze w tablicy haszującej opisującej wierzchołek traktowane są jako atrybuty danego wierzchołka. □

Zbiór testowo–treningowy przechowywany jest ze względu na swój rozmiar w postaci skompresowanej algorytmem *Lempel-Ziv 77* w czterech plikach: zdania źródłowe, zdania docelowe, lasy parsowania zdań docelowych oraz wyrównania zdań źródłowych i docelowych. Dane w tej postaci przetwarzane są przez część implementacji realizującej algorytm uczący z rozdziału 3.4, do postaci relacyjnej bazy danych w formacie *SQLite*.[2]

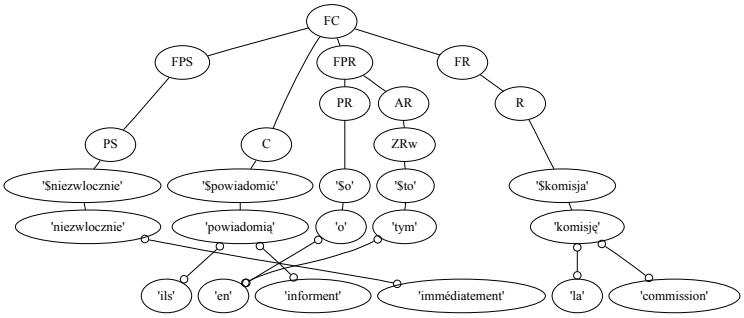
Baza danych zawiera *reguły tłumaczenia* pochodzące od grafów indukujących reguły, których wyznaczenie w grafach wyrównania jest celem algorytmu uczącego z rozdziału 3.4. Reguły są uproszczonymi opisami grafów indukujących reguły. Każda reguła jest strukturą złożoną z listy liści pewnego grafu indukującego regułę, korzenia grafu oraz listy opisującej pewną permutację liści tego grafu. Lista liści w każdej z reguł posortowana jest według kolejności wynikającej z porządku wyrazów w zdaniu źródłowym. Elementy tak posortowanej listy nazywać będziemy *kotwicami reguły*. Lista opisująca permutację kotwic reguły ma umożliwiać uporządkowanie kotwic w kolejności wynikającej z porządku wyrazów w zdaniu docelowym.

Przykład. Na rysunku 4.3(a) pokazano przykładowy graf indukujący regułę. Graf ten zostaje skonwertowany do następującej reguły:

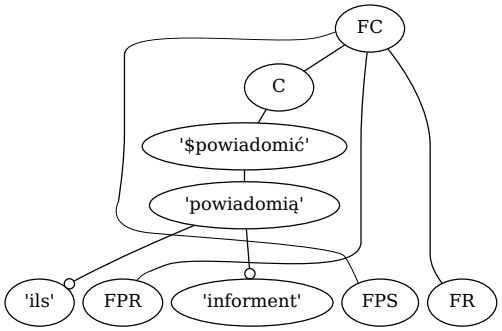
```
{ q|anchors|  => [ {q|category|=>q|'il'|, ...}, {q|category|=>q|'FPR'|, ...}
                 , {q|category|=>q|'informent'|, ...}, {q|category|=>q|'FPS'|, ...}
                 , {q|category|=>q|'FR'|, ...}
                 ]
, q|head|     => [ {q|category|=>q|'FC'|, ...}
, q|siblings| => [ 3, 'powiadomią', 1, 4 ]}
```

W powyższym zapisie dla większej czytelności pominięto większość atrybutów węzłów.

2. SQLite to osadzony system zarządzania bazą danych zgodny z postulatami ACID dostępny w postaci biblioteki programistycznej w języku C, powstały w roku 2000. Dystrybuowany jest na licencji typu *Public domain*. Oficjalna strona internetowa tego projektu to `http://sqlite.org`



(a) Przykładowy graf indukujący regułę.



(b) Zredukowany graf (a).

Rysunek 4.3. Przykładowy graf indukujący regułę i jego zredukowana, wskutek ekstrakcji reguł, postać.

Należy zauważyć, że graf pokazany na rysunku 4.3(a) zawiera podgrafy indukujące regułę. W trakcie przetwarzania podgrafy te zostaną skonwertowane do odpowiednich reguł, a oryginalny graf ulegnie redukcji do postaci przedstawionej na rysunku 4.3(b).

W wyniku przetworzenia zbioru testowo–treningowego opisanego w rozdziale 4.1 uzyskano ponad trzynaście tysięcy reguł; po usunięciu powtórzeń pozostało około dziesięciu tysięcy unikatowych reguł. Należy zaznaczyć, że plik bazy danych zawiera nie tylko same reguły, ale również pewną pewną ilość redundantnych danych koniecznych dla efektywnego zaimplementowania algorytmu tłumaczącego.

Bardzo istotną częścią implementacji realizującej algorytm tłumaczący opisany w rozdziale 3.5 jest kod wyszukujący w bazie danych reguł, które mogą zostać użyte do zbudowania grafu wyrównania ponad tłumaczonym zdaniem. Dla osiągnięcia satysfakcjonującej wydajności tej operacji baza reguł poddawana jest analizie i indeksowaniu.

Atrybuty reguł

W celu zmniejszenia ilość zapytań bazodanowych dla każdej reguły określane są cztery atrybuty binarne.

Atrybut	ustawiony, gdy
joinable-anchors	lista kotwic tej reguły jest prefiksem na liście kotwic innej reguły
joinable-head	głowa tej reguły jest pierwszą kotwicą innej reguły
expandable-head	głowa tej reguły jest jedyną kotwicą innej reguły
appendable-head	głowa tej reguły jest drugą lub kolejną kotwicą innej reguły

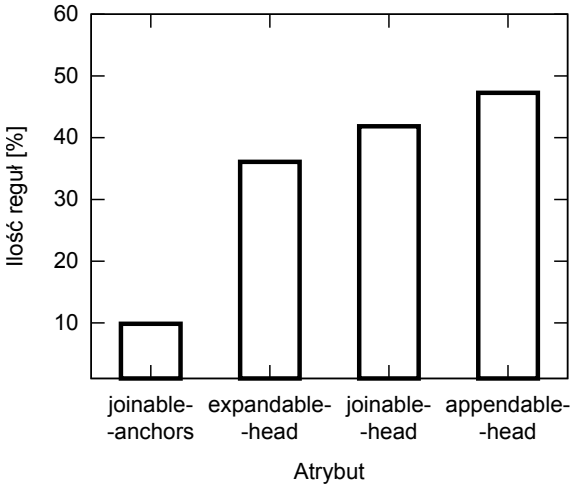
Atrybut *joinable-anchors* oznacza, że reguła, która została dopasowana do danego zestawu kotwic może nie być ich właściwą interpretacją. Sytuacja taka ma miejsce, gdy w bazie istnieje inna, „większa" reguła, której zbiór kotwic zawiera kotwice aktualnie rozpatrywane. Stosując regułę z ustawionym atrybutem *joinable-anchors* możemy narazić się na wybór nieoptymalnej ścieżki (zamiast jednego kroku, koniecznych będzie kilka, polegających na zastosowaniu kilku reguł, ale ostatecznie prowadzących do tego samego rezultatu) lub wręcz błędnej ścieżki (w przypadku gdyby nie zostały odnalezione inne reguły, których zastosowanie wraz z bieżącą, doprowadziłoby do tego samego rezultatu, co zastosowanie jednej większej reguły).

Atrybut *joinable-head* oznacza, że możliwe jest wyszukiwanie w bazie danych reguł pasujących od listy kotwic rozpoczynającej się od wierzchołka będącego głową bieżącej reguły. Podobnie — atrybut *appendable-head* oznacza, że ma sens powiększanie listy kotwic do wyszukiwania o głowę bieżącej reguły, gdyż być może uda się wyszukać pasującą regułę w bazie.

Atrybut *expandable-head* oznacza, że możliwe jest wyszukanie w bazie danych reguły pasującej do jednoelementowej listy kotwic, złożonej wyłącznie z wierzchołka będącego głową bieżącej reguły.

Pomiędzy atrybutami *expandable-head* a *joinable-head* zachodzi relacja zawierania, gdyż głowa reguły, która jest jedyną kotwiącą innej reguły, jest oczywiście pierwszą kotwicą owej reguły. Atrybuty *joinable-head* i *appendable-head* wykluczają się.

Rysunek 4.4 pokazuje procentowy udział każdego z atrybutów wśród reguł uzyskanych ze zbioru testowo–treningowego opisanego w rozdziale 4.1. W zaledwie 9,8% przypadków (ustawiony atrybut *joinable-anchors*) należy liczyć się z niebezpieczeństwem nieoptymalnego lub nieprawidłowego wyboru dokonanego w procesie konstruowania grafu wyrównania ponad tłumaczonym zdaniem. Obecność ustawionych pozostałych atrybutów w maksymalnie 47,3% reguł oznacza, że można zaniechać pewnych kwerend bazy danych w ponad połowie przypadków.



Rysunek 4.4. Ilość reguł uzyskanych ze zbioru treningowo–testowego, oznaczonych atrybutami ułatwiającymi wyszukiwanie

Przykład. Jedna spośród reguł z bazy danych reguł otrzymanych ze zbioru treningowo–testowego (rozdział 4.1), która ma ustawiony atrybut *joinable-anchors* to reguła:

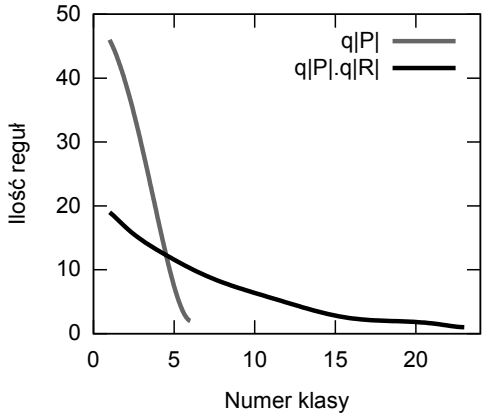
```
{ q|anchors|  => [ {q|category|=>q|'certaines'|, ...}]
, q|head|     => [ {q|category|=>q|'A'|, ...}
, q|siblings| => [ {q|category|=>q|'pewnych'|, ...} ]}
```

Atrybut ten jest ustawiony w przypadku tej reguły, gdyż wśród reguł znajduje się również następujące reguła:

```
{ q|anchors|  => [ {q|category|=>q|'certaines'|, ...}, {q|category|=>q|'procédures'|, ...}]
, q|head|     => [ {q|category|=>q|'N'|, ...}
, q|siblings| => [ {q|category|=>q|'odpowiednich'|, ...}
                 , {q|category|=>q|'procedur'|, ...}
                 ]}
```

Jeżeli w zdaniu, nad którym następuje budowanie grafu wyrównania, wystąpi podciąg „certaines procédures", wtedy podczas rozpatrywania jednoelementowej listy kotwic, złożonej z wierzchołka „certaines", zastosowanie pierwszej z reguł musi zostać opóźnione, aż do momentu, gdy rozpatrywana będzie dwuelementowa lista kotwic, złożona z wierzchołków „certaines" i „procédures". Wtedy to bowiem okaże się, że możliwe jest zaproponowanie tłumaczenia tego podciągu na ciąg „odpowiednich procedur", zamiast na ciąg „pewnych procedur" (o ile oczywiście w bazie reguł znalazłyby się reguły, które takie tłumaczenie mogłyby dostarczyć). □

Zmniejszenie liczby kwerend bazodanowych koniecznych dla wykonania pojedynczego kroku algorytmu tłumaczącego, którego liczba kroków wzrasta znacznie wraz z długością tłumaczonego zdania (złożoność rzędu $O(n^3)$, gdzie n to długość zdania), skutkuje istotnym zmniejszeniem czasu działania jego implementacji.



Rysunek 4.5. Ilość i wielkość klas na jakie można podzielić zbiór reguł o kategorii *ZP*

Indeks prefiksowy skrótów kotwic

Implementacja wyszukiwania bazodanowego w algorytmie tłumaczącym dąży do tego, aby dla danego zestawu kotwic znajdować idealne dopasowania. Jeżeli takie dopasowanie nie zostanie odnalezione, wtedy możliwe jest odszukanie reguł, które mają podobny do zadanego, choć nie identyczny, zestaw kotwic.

Reguły wyszukiwane są w bazie danych na podstawie tzw. *skrótów kotwic*. Skróty kotwic powstają poprzez zawężenie zbioru atrybutów związanych z każdym węzłem do maksymalnie czterech.

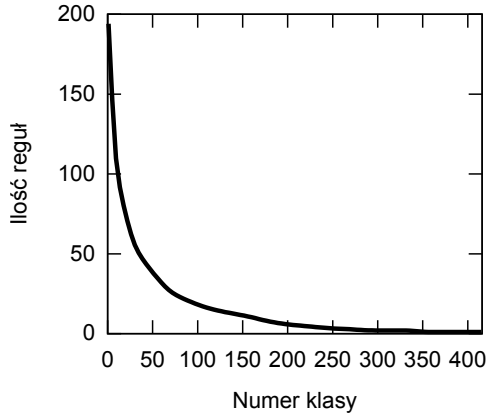
Przykład. Analizator składniowy systemu POLENG parsując zdania ze zbioru treningowo–testowego zaopatrzył każdy węzeł w średnio 15 atrybutów.

Najmniej atrybutów analizator składniowy określa dla węzłów oznaczonych kategorią `FRs`.

```
{ q|category| => q|FRs|, q|label| => q|#| }
```

Najwięcej atrybutów mają węzły kategorii `SFP`.

```
{ q|category| => q|SFP|, q|SuperHk| => 192629120, q|S| => q|person|, q|Sem| => q|nil|,
  q|R1| => q|nil|, q|Oblig1| => q|nil|, q|Symbol| => 182083224, q|Amb| => q|nil|,
  q|Ou| => q|nil|, q|Lu| => q|nil|, q|Przyim3| => q|nil|, q|Nr| => 3, q|Dop3| => q|nil|,
  q|Frag| => 641793, q|Ps4| => 16, q|Funkcja| => 641929, q|SpecPrzyim| => q|nil|,
  q|Split| => 22, q|Num| => 3, q|R2| => q|nil|, q|Skladnia| => 244014016, q|Sh| => 3,
  q|Stopien| => 384, q|Ps3| => 120, q|L2| => q|nil|, q|Ps2| => 2, q|PX| => 2, q|C| => 16,
  q|Oblig3| => q|nil|, q|Degree| => q|nil|, q|Kore| => q|nil|, q|L| => 182083224,
  q|Wyp2| => q|nil|, q|Podmiot| => 3, q|OczekiwanyP| => q|nil|, q|P| => 22, q|Rw| => q|nil|,
  q|Przyim| => q|nil|, q|T| => 192629368, q|Prep| => q|nil|, q|Ru| => q|nil|, q|O1| => q|nil|,
  q|Ps1| => 192629256, q|Przyim1| => 192629232, q|Przyim2| => q|nil|, q|Dop2| => 262051104,
  q|Prefix| => q|nil|, q|Typ| => q|nil|, q|SobiePrzyim| => 192629472, q|Ps5| => 192629224,
  q|label| => q|#|, q|Core| => 262051616, q|SNum| => q|nil|, q|Block| => 244014016,
  q|OczekiwanyPrzyim| => 25, q|Rp| => q|nil|, q|Gr| => q|nil|, q|SuperNr| => q|nil|,
  q|SpecWyp| => q|nil|, q|Neg| => q|nil|, q|Pytanie| => 192628360, q|Dop1| => 192629088,
  q|A| => 120, q|O| => q|nil|, q|Hk| => 192629336, q|L3| => q|nil|, q|R| => q|nil|,
  q|Zwrotnosc| => 25 }
```



Rysunek 4.6. Ilość i wielkość klas na jakie zostały podzielone reguły otrzymane ze zbioru treningowo–testowego.

Baza reguł jest analizowana w celu znalezienia takiego zestawu atrybutów, których wartości będą dzieliły cały zbiór reguł na klasy w miarę możliwości równe i jak najmniej liczne.

Przykład. Wśród zbioru reguł otrzymanych ze zbioru treningowo–testowego znalazło się 187 reguł, które w swoich kotwicach zawierają węzły kategorii `ZP`. Na rysunku 4.5 zaznaczono ilość i liczebność klas, jakie powstają, gdy za klucz do wyszukiwania reguł przyjąć wartość atrybutów *category* i P, oraz gdy klucz zostanie powiększony dodatkowo o wartości atrybutu R. W pierwszym przypadku otrzymujemy sześć klas, w drugim — dwadzieścia cztery.

Testowana implementacja wyznaczyła klucze indeksujące dla kotwic reguł pochodzących ze zbioru testowo–treningowego dzieląc zbiór reguł na 415 klas. Rozkład rozmiaru poszczególnych klas został pokazany na rysunku 4.6. Do klucza indeksującego zawsze zaliczona została wartość atrybutu *category*. W zależności od wartości tego atrybutu dodatkowo do klucza indeksującego dołączone zostały dodatkowe atrybuty.

Wartość *category*	Atrybuty dołączane do klucza indeksującego
`S`	*Ps1*
`PS`, `FPS`	*Typ*
`PR`	*P*, *Typ*
`FPR`, `P`, `FP`, `ZP`, `R`, `FR`	*P*, *R*
`FC`, `C`	*C*, *R*, a jeśli *R* jest nieustawiony wtedy zamiast niego *P* i *Przyim1*

Warto zauważyć, że w przypadku najliczniejszych klas (9 klas ma ponad 100 elementów) do zbudowania klucza indeksującego wykorzystane zostały tylko trzy atrybuty (przy założeniu, że dozwolone są cztery).

Przykład. Najliczniejszą klasą wyznaczoną według tego schematu jest klasa węzłów kategorii `FR`, dla których wartości atrybutów P i R to `dop` i `mno`. Klasa ta liczy 194 elementy. Jednak węzły w tej klasie nie zawsze mają określoną wartość danego atrybutu (np. atrybut *Dop1* określony jest tylko w 3% przypadków) lub wartości tych atrybutów nie pozwalają rozdzielenie klasy na mniejsze (np. atrybut *Skladnia* występuje we wszystkich węzłach w tej klasie, ale przyjmuje tylko dwie wartości — `nil` w 98,48% przypadków i `npr` w pozostałych 1,52%).

Po ustaleniu atrybutów kotwic, które mają wchodzić w skład klucza następuje generowanie indeksu do wyszukiwania reguł. Tworzonym indeksem jest standardowe B–drzewo, gdyż na tym typie indeksu możliwe jest wyszukiwanie prefiksowe, używane przez implementację algorytmu tłumaczącego celem zmniejszenia liczby kwerend bazodanowych.

Testowana implementacja buforuje wyniki przeszukiwania bazy reguł w celu ich późniejszego wykorzystania. Wśród reguł powstałych z korpusu treningowo–testowego zaobserwowano regułę, która nakazują tłumaczenie francuskiej frazy „il est" jako polskiego „jest", oraz regułę, która tłumaczy frazę frazy „il est apparu" jako „wskazano". Gdy w pewnym kroku algorytmu tłumaczącego konieczne jest wyszukanie reguł pasujących do klucza `il est`, wtedy z bazy reguł pobrane zostaną obie wspomniane wyżej reguły.

4.3. Badanie własności implementacji

4.3.1. Zdolność pozyskiwania reguł

Korpus treningowo–testowy został przeanalizowany przez algorytm uczący w ciągu około 40 sekund na komputerze klasy netbook (procesor taktowany zegarem 1,6GHz, całkowity rozmiar pamięci RAM – 512MB, system operacyjny z rodziny linuks – Ubuntu 10.04).

Uzyskano 14176 reguł. Z jednego zdania korpusu uzyskano średnio 25 reguł. Po odrzuceniu powtórzeń pozostało 11026 reguł. Wśród powtórzeń najczęstsze były reguły pochodzące ze zdań zawierających liczby np. sygnatury akt.

Przykład. Najwięcej, bo aż 108 reguł, uzyskano z pary zdań:

	PL	*FR*
133	” udział kapitałowy ” oznacza udział kapitałowy w rozumieniu art. 17 zdanie pierwsze czwartej dyrektywy Rady 78/660/EWG z dnia 25 lipca 1978 r. wydanej na podstawie art. 54 ust. 3 lit. g) Traktatu w sprawie rocznych sprawozdań finansowych niektórych rodzajów spółek16 , lub bezpośredniej i pośredniej własności 20 % lub więcej praw głosu lub kapitału przedsiębiorstwa ;	” participation ” : une participation au sens de l’ article 17 , première phrase , de la quatrième directive 78/660/CEE du Conseil du 25 juillet 1978 concernant les comptes annuels de certaines formes de sociétés (16) ou le fait de détenir , dire ctement ou indirectement , au moins 20 % des droits de vote ou du capital d’ une entreprise ;

Jedną z reguł jest reguła rozpoznająca napis „du 25 juillet 1978” i tłumacząca go wprost na frazę rzeczownikową „z dnia 25 lipca 1978 r.”. □

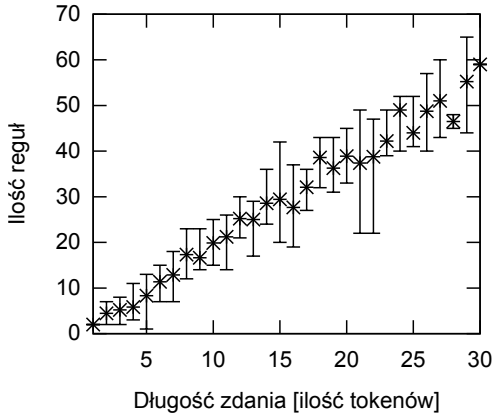
Z części korpusu, która zawiera jedynie litery i znaki interpunkcyjne (jest to 64% korpusu) uzyskano 8810 reguł. W tym przypadku średnia liczba reguł uzyskanych z jednego zdania to 23. Po odrzuceniu duplikatów pozostało 2281 reguł.

Wniosek. Algorytm uczący poprawnie przetwarza liczby, daty, wartości procentowe, monetarne itp., jednak generuje przy tym bardzo szczegółowe reguły (tłumaczące konkretne wartości). Spowodowane jest to po części użytym parserem, który posiada funkcjonalność *named entities recognition*[3] [14].

Należy spodziewać się, że bez rozszerzenia algorytmu tłumaczącego o rozpoznawanie *named entities*, algorytm tłumaczący, w przypadku napotkania np. wartości liczbowej, której nie zaobserwowano w procesie uczenia, będzie przetwarzał zdania segmentami, a to skutkować może gorszą jakością tłumaczenia.

Wydaje się, że zbiór reguł powinien zostać oczyszczony z reguł przetwarzających wartości liczbowe. Algorytm uczący może pozyskać z korpusu bardzo dużo reguł tego typu, które – jak należy przypuszczać, ze względu na ich specyfikę, będą rzadko wykorzystywane.

3. *Named entity recognition* jest zadaniem z dziedziny pozyskiwania informacji, polegającym na lokalizowaniu i klasyfikacji jednostek tekstu, które w zależności od zastosowania, powinny być przetwarzane jako atomowe. Jednostkami podlegającymi identyfikacji mogą być nazwy osób, organizacji lub miejsc, wyrażenia określające czas lub ilość, itp.



Rysunek 4.7. Średnia ilość reguł w zależności od długości zdania

Na rysunku 4.7 pokazana została zależność ilości reguł uzyskiwanych z pojedynczego grafu wyrównania od długości zdania docelowego. Średnia ilość reguł rośnie liniowo wraz ze wzrostem długości zdania, jednak odchylenia od średniej są znaczne.

Wniosek. Ilość reguł pozyskiwanych z pojedynczego grafu wyrównania zależy nie tylko od długości zdania, ale również w dużym stopniu od struktury zdania docelowego (głębokości drzewa parsowania) i wyrównania pomiędzy zdaniem źródłowym a docelowym.

4.3.2. Zdolność rekonstrukcji danych treningowych

Poprawność implementacji algorytmów opisanych w rozdziale 3, a więc również samych algorytmów, zweryfikowana została przez określenie czy i w jakim stopniu algorytm tłumaczący z rozdziału 3.5 jest zdolny budować grafy wyrównania (a przez to konstruować tłumaczenia) z reguł uzyskanych przy użyciu algorytmu uczącego z rozdziału 3.4.

Eksperyment został przeprowadzony z użyciem korpusu treningowo–testowego. Baza reguł, która powstała w wyniku wykonania algorytmu uczącego, została w całości udostępniona algorytmowi tłumaczącemu. Część francuska tego samego korpusu została użyta jako zdania do przetłumaczenia.

Korzystając z bazy 2841 reguł implementacja algorytmu tłumaczącego zdołała poprawnie zrekonstruować graf wyrównania w 34% przypadków. Dokładna analiza otrzymanych wyników wykazała, że jedną z przyczyn niskiego stopnia rekonstrukcji danych treningowych jest utrata części informacji w procesie uczenia.

Problem dotyczy przypadków, gdy w zdaniu źródłowym występują tokeny, które nie mają określonego odpowiednika po stronie docelowej. Wystąpienie takiego tokenu w zdaniu źródłowym powoduje, że zdanie przetwarzane jest rozdzielnymi segmentami.

Przykład. Jedną z przyczyn pojawiania się niewyrównanych tokenów są różnice w interpunkcji w zdaniu źródłowym i docelowym.

	PL	*FR*
3	Jeśli konstrukcyjna prędkość maksymalna pojazdu jest niższa od wymaganej na potrzeby badania , badanie musi być wykonane przy szybkości maksymalnej pojazdu ;	Lorsque , par construction , la vitesse maximale du véhicule est inférieure à celle prescrite pour un essai , l' essai doit être fait à la vitesse maximale du véhicule ;

W powyższej parze zdań pierwsze dwa przecinki w zdaniu francuskim nie mają swoich odpowiedników po stronie polskiej.

W korpusie treningowo–testowym znajduje się 33% par zdań, w których występuje co najmniej jeden token źródłowy, który nie jest wyrównany do żadnego tokenu docelowego.

Wniosek. Warunek eksperymentu polegający na uznaniu próby za zaliczoną na podstawie idealnego odtworzenia grafu wyrównania, ogranicza maksymalną możliwą skuteczność implementacji do 67%.

W związku z wysoką czułością implementacji na „zakłócenia" w zdaniu źródłowym, wprowadzono następujące modyfikacje do implementacji algorytmu uczącego i tłumaczącego. W procesie analizy grafów wyrównania tworzone są reguły specjalne, przechowujące informację o tym, że pewien token może zostać pominięty w procesie tłumaczenia. W implementacji algorytmu tłumaczącego dopuszczono łączenie dowolnej reguły z regułą specjalną, która nie generuje żadnego węzła w grafie wyrównania.

Eksperyment został powtórzony z użyciem zmodyfikowanej implementacji. Osiągnięto wynik 45% poprawnych rekonstrukcji.

Analiza wyników ujawniła anomalie w korpusie użytym podczas eksperymentu, wynikające z różnic w podziale zdania docelowego na tokeny pomiędzy podziałem użytym podczas określania wyrównania zdania źródłowego i docelowego, a podziałem stosowanym wewnętrznie przez system Poleng.

Przykład. Różnice w tokenizacji przyjętej w korpusie DGT–TM, którego podzbiorem jest korpus treningowo–testowy, a tokenizacją systemu POLENG objawiają się najczęściej w przypadku znaków interpunkcyjnych oraz liczb, symboli lub oznaczeń.

	PL	*FR*
51	Ilości (pozycje 19-26) poniżej połowy jednostki należy zapisywać jako „ 1 ” .	Les quantités (octets 19 à 26) inférieures à la moitié d' une unité devraient être enregistrées de la manière suivante : 1 .

W przypadku polskiego zdania z powyższej pary zdań obserwujemy brak tokenów `(`, `)`, `"` i `.` jako liści w drzewie parsowania.

Różnica w tokenizacji występuje w przypadku 31% zdań użytego korpusu.

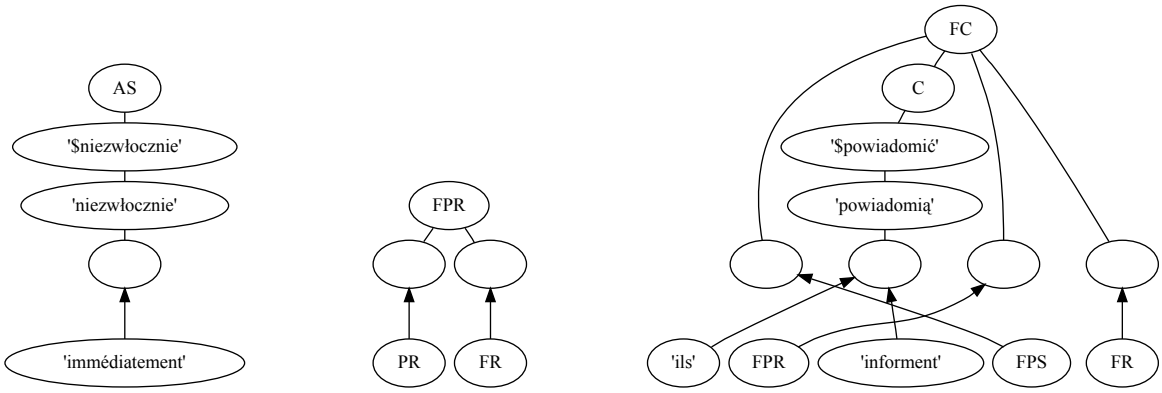
Wniosek. Przy restrykcyjnym kryterium zaliczenia próby jako udanej (pełna rekonstrukcja grafu wyrównania użytego podczas uczenia) anomalie występujące w korpusie znacznie wpływają na obserwowaną sprawność implementacji.

Ponieważ anomalie powstałe w trakcie przygotowania korpusu nie są błędem testowanej implementacji algorytmów z rozdziału 3, eksperyment został powtórzony z ograniczonym korpusem zdań. W celu „odszumienia” wyniku użyte zostały tylko zdania, w których nie występował problem niezgodności tokenizacji. Dla korpusu ograniczonego do 390 par zdań osiągnięto pełną zgodność rekonstruowanych grafów wyrównań z oryginałami użytymi w etapie uczenia.

Wniosek. Sprawność systemu mierzona zdolnością rekonstrukcji danych treningowych zależy silnie od jakości tych danych. W przypadku oczyszczonych danych należy spodziewać się sprawności bliskiej lub równiej 100%.

4.3.3. Zdolność generowania nowych grafów wyrównania

Celem bieżącego eksperymentu jest określenie zdolności algorytmu tłumaczącego do generowania grafu wyrównania nad zdaniem, które nie zostało użyte w procesie pozyskiwania reguł. Dla zminimalizowania zakłóceń w wynikach, na potrzeby tego eksperymentu posłużono się ograniczonym korpusem treningowo–testowym, dla którego testowana implementacja została zweryfikowana jako poprawna (patrz: rozdział 4.3.2).



(a) słownikowa (b) parsowania (c) mieszana

Rysunek 4.8. Przykładowe reguły tłumaczenia

W wyniku analizy ograniczonego korpusu testowo–treningowego uzyskano 7821 reguł, z czego 2154 unikatowych. Na rysunku 4.8 pokazano przykłady otrzymanych reguł.

Reguła słownikowa to taka reguła, która rozpoznaje token lub tokeny źródłowe i dostarcza możliwe tłumaczenie, w postaci jednego lub więcej tokenu docelowego i informacji gramatycznej z nim związanej.

Reguła parsowania to taka reguła, która zawiera wiedzę pochodzącą z parsera użytego do oznaczenia składniowego zdań docelowych korpusu. Kotwicami reguł tego typu są wierzchołki parsowania, a w wyniku zastosowania reguły tego typu nie może ulegać zmianie kolejność kotwic.

Reguła mieszana to taka reguła, której kotwicami mogą być zarówno wierzchołki źródłowe jaki i wierzchołki parsowania. Do tej klasy zaliczać będziemy reguły, które nie mogą zostać sklasyfikowane jako reguły parsowania, ze względu na zamianę kolejności kotwic.

W bieżącym eksperymencie do tłumaczenia każdego ze zdań ograniczonego korpusu treningowo–testowego wykorzystana zostanie, różna dla każdego tłumaczonego zdania, część bazy reguł. W tym celu każda z pozyskanych reguł została oznaczona identyfikatorem pary zdań, z której została otrzymana. Ponieważ baza reguł została utworzona z całego korpusu treningowo–testowego, w tłumaczeniu każdego ze zdań tego korpusu nie zostaną wykorzystane reguły typu mieszanego pochodzące z aktualnie tłumaczonego zdania.

Wykorzystanie w eksperymencie reguł słownikowych i reguł parsowania pochodzących z grafu wyrównania aktualnie tłumaczonego zdania jest dopuszczalne i uzasadnione. Reguły słownikowe są konieczne, gdyż żaden system tłumaczący nie jest w stanie przetłumaczyć słów, dla których nie zna odpowiedników. Ze względu na rozmiar użytego w bieżącym eksperymencie korpusu wysoce prawdopodobnym jest, że część prób nie powiodłaby się z powodu braku odpowiednika słowa źródłowego. Tymczasem określenie zdolności pozyskiwania informacji słownikowej z korpusu użytego w etapie uczenia nie jest przedmiotem bieżącego eksperymentu. Podobnie rzecz się ma z regułami parsowania. Wysoce prawdopodobne jest, że ze względu na rozmiar użytego korpusu wśród reguł zabrakłoby tych, które opisują składnię języka docelowego. Ponadto należy dodać, że baza reguł w każdym przypadku może zostać łatwo rozszerzona o reguły parsowania języka docelowego (np. w wyniku przeprowadzenia analizy składniowej monojęzycznego korpusu tekstów, co nie stoi w sprzeczności z założeniami przyjętymi dla algorytmów opisanych w rozdziale 3; przykładem takiego korpusu może być zbiór artykułów zamieszczonych na stronach internetowych wydawców prasy o zasięgu ogólnokrajowym).

W wyniku przeprowadzenia eksperymentu z zachowaniem opisanych powyżej warunków stwierdzono 61% skuteczność implementacji w generowaniu pełnych grafów wyrównania nieobserwowanych wcześniej zdań.

Wniosek. W przypadku kiedy algorytm tłumaczący nie mógł wygenerować pełnego grafu wyrównania, zdanie efektywnie zostało przetworzone w segmentach. Jest wysoce prawdopodobne, że część powstałych w ten sposób polskich odpowiedników zdań francuskich tłumaczeń mogłaby zostać uznana za poprawne tłumaczenie.

Wśród reguł mieszanych znajdują się reguły, które przypominają reguły parsowania, jednak łamią zasadę niezmieniania kolejności kotwic. Można powiedzieć, że reguły tego typu odzwierciedlają proste zależności składniowe pomiędzy językiem źródłowym, a docelowym. Dopuszczając wykorzystanie także tego typu reguł skuteczność implementacji w generowaniu pełnych grafów wyrównań wzrasta do poziomu 71%.

Rozdział 5

Podsumowanie

Algorytmy opisane w rozdziale 3 zostały zweryfikowane praktycznie podczas eksperymentów przeprowadzonych na ich implementacji, co opisano w rozdziale 4.

Wyniki przeprowadzonych eksperymentów wskazują na to, że testowane algorytmy realizują praktyczny cel, dla którego zostały zaprojektowane — z niewielkiego korpusu dwujęzycznego oznaczonego składniowo po stronie polskiej, pozyskiwane są reguły, dzięki którym można konstruować tłumaczenia na język polski.

Opisane w rozdziale 3 algorytmy mogłyby znaleźć zastosowanie w systemie tłumaczenia tekstów technicznych, na przykład instrukcji obsługi urządzeń. Taki system, któremu na etapie trenowania przedstawionoby tekst instrukcji obsługi pewnego urządzenia elektronicznego np. routera bezprzewodowego, powinien być w stanie przetłumaczyć z zadowalającą jakością instrukcję obsługi podobnego urządzenia np. modelu pochodzącego od innego producenta.

System tłumaczący zbudowany na bazie algorytmów z rozdziału 3 posiadałby również tą cechę, że mógłby być trenowany przyrostowo. Baza reguł może zostać zainicjowana jedynie regułami parsowania i słownikowymi – pierwsze można uzyskać wprost z drzew parsowania dowolnego tekstu w języku polskim; drugie mogą pochodzić z dowolnego słownika dwujęzycznego. System z taką bazą reguł tłumaczyłby kolejno zdania z pewnego tekstu w języku obcym, a operator systemu, w przypadku gdy tłumaczenie byłoby niezadowalającej jakości, podawałby poprawne tłumaczenie i wyrównywał je z dokładnością co do wyrazu do oryginalnego zdania. Następnie wyrównana para zdań służyłaby za źródło nowych reguł dla systemu.

Algorytmy przedstawione w rozdziale 3 mogą być potraktowane jako próba zamodelowania procesu uczenia się języka obcego przez człowieka. Mogłyby więc zostać zastosowane do porównania podręczników do nauki języka obcego. Każdy z badanych podręczników, a konkretnie zbiór podawanych w nim przykładów (wraz z ich tłumaczeniami na język polski), należałoby potraktować jako osobny korpus treningowy.

Wynikiem porównania otrzymanych zbiorów reguł mogłoby wskazanie „lepszego" podręcznika. Zbiór reguł, który zawierałby więcej unikatowych reguł, a szczególnie typu mieszanego, niż zbiory powstałe z innych podręczników, oznaczałby, że podręcznik, na bazie którego powstawał ten zbiór, szczególnie dobrze eksponuje różnice pomiędzy językiem obcym a ojczystym (językiem polskim).

Bibliografia

[1] *The DGT Multilingual Translation Memory of the Acquis Communautaire: DGT-TM.* `http://langtech.jrc.it/DGT-TM.html`.

[2] Yaser Al-Onaizan, Ulrich Germann, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Daniel Marcu, Kenji Yamada. Translating with scarce resources. *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, strony 672–678. AAAI Press / The MIT Press, 2000.

[3] Björn Beskow. Unification-based transfer in machine translation. Raport instytutowy RULL #24, Wydział Lingwistyki Uniwersytetu w Uppsala, Szwecja, 1993.

[4] Thomas G. Bever. The cognitive basis for linguistic structures. J. R. Hayes, redaktor, *Cognition and the Development of Language*, strony 279–352. Wiley, New York, USA, 1970.

[5] David Crystal, redaktor. *The Cambridge Encyclopedia of Language*. Cambridge University Press, wydanie 2, 1997.

[6] Elżbieta Dobryjanowicz. *Podstawy przetwarzania języka naturalnego: wybrane metody analizy składniowej*. Akademicka Oficyna Wydawnicza RM, 1992.

[7] Bonnie J. Dorr, Pamela W. Jordan, John W. Benoit. A survey of current paradigms in machine translation. M. Zelkowitz, redaktor, *Advances in Computers*, wolumen 49, strony 1–68. Academic Press, 1999.

[8] Heidi J. Fox. Phrasal cohesion and statistical machine translation. *Proceedings of EMNLP-02*, strony 304–311, 2002.

[9] Michel Galley, Mark Hopkins, Kevin Knight, Daniel Marcu. What's in a translation rule? *Proceedings of the Human Language Technology and North American Association for Computational Linguistics Conference (HLT/NAACL-04)*, Boston, USA, 2004.

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

[11] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. *Wprowadzenie do teorii automatów, języków i obliczeń*. Wydawnictwo Naukowe PWN, 2005.

[12] Krzysztof Jassem. *Przetwarzanie tekstów polskich w systemie tłumaczenia automatecznego POLENG*. Wydawnictwo Naukowe UAM, Poznań, 2006.

[13] Krzysztof Jassem, Filip Gralinski, Tomasz Kowalski. Applying transition networks in translating polish e-mails. Mieczyslaw A. Klopotek, Slawomir T. Wierzchon, Krzysztof Trojanowski, redaktorzy, *IIS*, Advances in Soft Computing, strony 521–527. Springer, 2003.

[14] Krzysztof Jassem, Tomasz Kowalski. Narzędzia do opisu i interpretacji skończonych sieci przejść w systemie poleng. *Speech and Language Technology. Technologia Mowy i Języka*, 7, 2003.

[15] Krzysztof Jassem, Tomasz Kowalski. An algorithm for extracting translation rules from scarce bilingual corpora. *Proceedings of 1th International Multiconference on Computer Science and Information Technology*, Wisła, 2006.

[16] Marcin Junczys-Dowmunt. It's all about the trees – towards a hybrid syntax-based mt system. *Proceedings of 4th International Multiconference on Computer Science and Information Technology*, strony 219–226, Mrągowo, 2009.

[17] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010.

[18] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, Evan Herbst. Moses: Open source toolkit for statistical machine translation. *ACL*. The Association for Computer Linguistics, 2007.

[19] Tomasz Kowalski, Krzysztof Jassem. Machine translation using scarce bilingual corpora. wolumen 11/1–2. TASK, Gdańsk, 2007.

[20] Jadwiga Linde-Usienkiewicz, redaktor. *Wielki słownik angielsko–polski*. Wydawnictwo Naukowe PWN, 2002.

[21] Franz Josef Och, Hermann Ney. Improved statistical alignment models. *In Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, strony 440–447, 2000.

[22] Franz Josef Och, Hermann Ney, Franz Josef, Och Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29, 2003.

[23] Karolina Owczarzak, Josef van Genabith, Andy Way. Labelled dependencies in machine translation evaluation. *ACL Workshop on Statistical Machine Translation*, Praga, Czechy, 2007.

[24] Fernando C. N. Pereira, David H. D. Warren. Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. *Artif. Intell.*, strony 231–278, 1980.

[25] Adam Przepiórkowski, Aleksander Buczyński. Shallow parsing and disambiguation engine. *Proceedings of the 3rd Language & Technology Conference*, Poznań, 2007.

[26] Ralf Steinberger, Bruno Pouliquen, Anna Widiger, Camelia Ignat, Tomaz Erjavec, Dan Tufis, Daniel Varga. The jrc-acquis: A multilingual aligned parallel corpus with 20+ languages. *CoRR*, abs/cs/0609058, 2006. informal publication.

[27] Berment Vincent. *Méthodes pour informatiser des langues et des groupes de langues « peu dotées »*. Praca doktorska, Grenoble University, Francja, 2004.

Indeks

Spis rysunków