Uniwersytet im. Adama Mickiewicza
Wydział Matematyki i Informatyki

Paweł Skórzewski
nr albumu: 301654

Efektywny parsing języka naturalnego przy użyciu gramatyk probabilistycznych

Praca magisterska na kierunku:
informatyka

Promotor:
dr hab. Krzysztof Jassem

Poznań 2010

Oświadczenie

Poznań, dnia .....

Ja, niżej podpisany Paweł Skórzewski, student Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu, oświadczam, że przedkładaną pracę dyplomową pt. *Efektywny parsing języka naturalnego przy użyciu gramatyk probabilistycznych* napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania rozprawy lub jej części innym osobom ani nie odpisywałem tej rozprawy lub jej części od innych osób.

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

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

.....

Spis treści

Oświadczenie 1

Spis oznaczeń 4

Rozdział 1. Wstęp 5

1.1. O tłumaczeniu automatycznym i probabilistycznych gramatykach bezkontekstowych 5

1.2. Cel i założenia niniejszej pracy 5

Rozdział 2. Podstawy teoretyczne 7

2.1. Pojęcia związane z rachunkiem prawdopodobieństwa 7

2.2. Pojęcia związane z teorią grafów 8

2.3. Gramatyki i języki formalne 12

2.3.1. Symbole, alfabety i łańcuchy 12

2.3.2. Języki 13

2.3.3. Gramatyki formalne 14

2.3.4. Probabilistyczne gramatyki bezkontekstowe 20

Rozdział 3. Przedstawienie istniejących algorytmów parsingu probabilistycznych gramatyk bezkontekstowych 27

3.1. Ogólna charakterystyka podstawowych metod parsingu gramatyk bezkontekstowych 27

3.1.1. Parsing zstępujący i wstępujący 27

3.1.2. Programowanie dynamiczne i parsing tablicowy 28

3.2. Algorytm Earleya 30

3.2.1. Algorytm Earleya dla gramatyk bezkontekstowych 30

3.2.2. Algorytm Earleya dla probabilistycznych gramatyk bezkontekstowych 32

3.3. Algorytm CYK 36

3.3.1. Algorytm CYK dla gramatyk bezkontekstowych 36

3.3.2. Algorytm CYK dla probabilistycznych gramatyk bezkontekstowych 37

3.4. Algorytmy parsowania wykorzystujące metody z algorytmów grafowych 40

3.4.1. Wykorzystanie algorytmów grafowych do parsowania 40

3.4.2. Strategia przeszukiwania „najpierw najlepszy” 41

3.4.3. Strategia przeszukiwania wiązkowego 42

Rozdział 4. Algorytm A* w parsingu — opis i porównanie różnych rodzajów heurystyk 43

4.1. Algorytm A* — opis działania w ogólnym przypadku 43

4.2. Algorytm A* w parsingu 49

4.2.1. Ogólny opis działania algorytmu parsingu A* 49

4.2.2. Heurystyki oparte na oszacowaniach kontekstu 50

4.2.3. Heurystyki oparte na rzutowaniu gramatyki 51

4.2.4. Przykład parsingu za pomocą algorytmu A* 53

Rozdział 5. Implementacja algorytmu A* w systemie Translatica, uzasadnienie dokonanych wyborów 60
5.1. Rozwiązania wykorzystywane w systemie Translatica 60
5.1.1. System tłumaczenia automatycznego Translatica 60
5.1.2. Algorytm parsingu systemu Translatica 60
5.1.3. Gramatyki atrybutowe 61
5.1.4. Wykorzystanie maszyny stosowej w parserze 63
5.2. Implementacja algorytmu A* w parserze systemu Translatica 63
5.2.1. Implementacja algorytmu A* z heurystyką SX 63
5.2.2. Rzutowanie gramatyki atrybutowej 65
5.2.3. Implementacja algorytmu A* z heurystyką NULL 66
Rozdział 6. Wyniki i wnioski 67
6.1. Porównanie dotychczasowego parsera i parsera wykorzystującego algorytm A* pod względem poprawności i efektywności 67
6.1.1. Przeprowadzone testy 67
6.1.2. Szybkość tłumaczenia 67
6.1.3. Jakość tłumaczenia 68
6.2. Ocena przydatności zastosowanych rozwiązań i perspektywy na przyszłość 69
Rozdział 7. Podsumowanie 71
Bibliografia 72

Spis oznaczeń

A, B, C, D	symbole pomocnicze
$\mathcal{A}$	zbiór atrybutów
E	zbiór krawędzi grafu
$\mathcal{E}$	rodzina zbiorów wyrażeń atrybutowych
$\mathcal{F}$	σ-algebra
G	gramatyka
O	notacja asymptotyczna
P	prawdopodobieństwo reguły
$\mathbf{P}$	prawdopodobieństwo
$\mathcal{P}$	zbiór potęgowy
R	zbiór reguł produkcji
$\mathbb{R}$	zbiór liczb rzeczywistych
S	symbol początkowy
T	zbiór symboli końcowych
U	zbiór wierzchołków grafu
V	zbiór symboli pomocniczych; alfabet
$\mathcal{V}$	rodzina zbiorów wartości atrybutów
W	waga krawędzi grafu
X, Y, Z	symbole końcowe bądź pomocnicze
a, b, c	symbole końcowe
$\mathbf{a}, \mathbf{b}, \mathbf{c}$	atrybuty
d	odległość w grafie
e	krawędzie grafu
i, j, k, l, m, n	liczby
r	reguły produkcji
u, v, w	łańcuchy symboli końcowych
u, v, x, y	wierzchołki grafu
$\mathbf{x}, \mathbf{y}, \mathbf{z}$	wartości atrybutów
$w_1, w_2, \ldots$	symbole końcowe
Γ, Δ	grafy, drzewa
Ω	przestrzeń zdarzeń elementarnych
α	prawdopodobieństwo zewnętrzne
β	prawdopodobieństwo wewnętrzne
ϵ	łańcuch pusty
ζ, ξ, ω	łańcuchy symboli pomocniczych i końcowych
ν	prawdopodobieństwo Viterbiego
$\emptyset$	zbiór pusty
log	logarytm (binarny, jeśli nie podano inaczej)

Rozdział 1

Wstęp

1.1. O tłumaczeniu automatycznym i probabilistycznych gramatykach bezkontekstowych

Kwestia usprawnienia komunikacji między użytkownikami różnych języków zaprzątała umysły ludzkie od najdawniejszych czasów. Już w XVII w. Leibniz i Kartezjusz głosili koncepcje języków uniwersalnych, za pomocą których można by było wyrazić rozmaite pojęcia w sposób ścisły i jednoznaczny — użycie takiego języka jako pośredniego uprościłoby proces przekładu myśli z jednego języka na inny. Pierwsze próby wynalezienia maszyn, które tłumaczyłyby zdania w sposób automatyczny, pojawiły się już w latach 30. XX wieku, jednak dopiero wynalazek komputerów sprawił, że realizacja takich pomysłów stała się naprawdę możliwa.

Powstanie i rozwój teorii gramatyk kombinatorycznych, a w szczególności gramatyk bezkontekstowych, które okazały się dobrym narzędziem do formalnego opisu języków zarówno sztucznych, jak i naturalnych, miały znaczący wpływ na postępy w dziedzinie tłumaczenia automatycznego. Klasyczne algorytmy parsingu (czyli analizy składniowej) zdań, takie jak algorytm CYK czy parser Earleya, które korzystają z formalizmu gramatyk bezkontekstowych, są powszechnie wykorzystywane do tłumaczenia maszynowego.

Naturalnym rozwinięciem gramatyk bezkontekstowych są gramatyki probabilistyczne, które korzystają narzędzi rachunku prawdopodobieństwa, aby lepiej modelować statystyczne cechy języka. Możliwość wykorzystania dodatkowej wiedzy o języku, jaką oferują probabilistyczne gramatyki bezkontekstowe, otwiera nowe perspektywy dla tłumaczenia automatycznego.

1.2. Cel i założenia niniejszej pracy

Celem mojej pracy magisterskiej jest przedstawienie metod parsingu probabilistycznych gramatyk bezkontekstowych, a w szczególności algorytmu A*. Praca zawiera również opis implementacji tego algorytmu w systemie tłumaczenia automatycznego Translatica, będącej przedmiotem projektu magisterskiego.

W rozdziale 2 przedstawione są wiadomości z rachunku prawdopodobieństwa, teorii grafów i teorii języków formalnych, które są potrzebne w dalszej części pracy. W szczególności przedstawione są najważniejsze fakty dotyczące gramatyk bezkontekstowych i probabilistycznych gramatyk bezkontekstowych.

Rozdział 3 prezentuje przegląd istniejących algorytmów parsingu probabilistycznych gramatyk bezkontekstowych.

W rozdziale 4 omówiony jest algorytm A* oraz jego zastosowanie do parsowania.

Rozdział 5 przedstawia system tłumaczenia automatycznego Translatica i opisuje, w jaki sposób algorytm parsingu A* został zaimplementowany w tym systemie w ramach projektu powiązanego z niniejszą pracą magisterską.

Wyniki uzyskane w projekcie i płynące z nich wnioski zawarte są w rozdziale 6.

Krótkim podsumowaniem całej pracy jest rozdział 7.

Rozdział 2

Podstawy teoretyczne

2.1. Pojęcia związane z rachunkiem prawdopodobieństwa[1]

Definicja 2.1 (σ-algebra)**.** Niech Ω będzie zbiorem. *σ-algebrą* nazywamy niepustą rodzinę $\mathcal{F}$ podzbiorów zbioru Ω spełniającą następujące warunki:

(1) $\Omega \in \mathcal{F}$,
(2) jeżeli $A_1, A_2, \ldots \in \mathcal{F}$, to $\bigcup_{n=1}^{\infty} A_n \in \mathcal{F}$,
(3) jeżeli $A \in \mathcal{F}$, to $\Omega \setminus A \in \mathcal{F}$.

Twierdzenie 2.1 (własności σ-algebr)**.** *Niech $\mathcal{F}$ będzie σ-algebrą podzbiorów zbioru Ω. Wówczas zachodzą następujące własności:*

(1) $\emptyset \in \mathcal{F}$.
(2) Jeżeli $A_1, A_2, \ldots \in \mathcal{F}$, to $\bigcap_{n=1}^{\infty} A_n \in \mathcal{F}$.
(3) Jeżeli $A, B \in \mathcal{F}$, to $A \setminus B \in \mathcal{F}$.

Można powiedzieć, że σ-algebra jest rodziną zbiorów zamkniętą na skończone i przeliczalne operacje teoriomnogościowe.

W szczególności, zbiór $\mathcal{P}(\Omega)$ wszystkich podzbiorów zbioru Ω (*zbiór potęgowy*) jest σ-algebrą. Jest to największa σ-algebra podzbiorów zbioru Ω.

Definicja 2.2 (przestrzeń probabilistyczna)**.** *Przestrzenią probabilistyczną* nazywamy trójkę $(\Omega, \mathcal{F}, \mathbf{P})$, gdzie:

— Ω jest dowolnym zbiorem, nazywanym *przestrzenią zdarzeń elementarnych*,
— $\mathcal{F}$ jest pewną σ-algebrą podzbiorów zbioru Ω, podzbiory te nazywamy *zdarzeniami*,
— $\mathbf{P}\colon \mathcal{F} \to \mathbb{R}$ jest funkcją spełniającą następujące aksjomaty:
(1) $\mathbf{P}(A) \geqslant 0$ dla każdego $A \in \mathcal{F}$,
(2) $\mathbf{P}(\Omega) = 1$,
(3) dla ciągu (A_n) parami rozłącznych zdarzeń z $\mathcal{F}$ (tj. $A_i \cap A_j = \emptyset$ dla $i \neq j$) zachodzi równość

$$\mathbf{P}\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} \mathbf{P}\left(A_n\right).$$

Funkcję $\mathbf{P}$ nazywa się *prawdopodobieństwem.*

Twierdzenie 2.2 (własności prawdopodobieństwa)**.** *Niech $(\Omega, \mathcal{F}, \mathbf{P})$ będzie przestrzenią probabilistyczną. Wówczas zachodzą następujące własności:*

(1) $\mathbf{P}(\emptyset) = 0$.
(2) Dla każdego $A \in \mathcal{F}$ zachodzi $0 \leqslant \mathbf{P}(A) \leqslant 1$.

[1] Na podstawie [9].

(3) $\mathbf{P}(\Omega \setminus A) = 1 - \mathbf{P}(A)$ *dla* $A \in \mathcal{F}$.

Definicja 2.3 (prawdopodobieństwo warunkowe)**.** Niech $(\Omega, \mathcal{F}, \mathbf{P})$ będzie przestrzenią probabilistyczną. Jeżeli $A, B \in \mathcal{F}$, $\mathbf{P}(B) \neq 0$, to *prawdopodobieństwo warunkowe* zdarzenia A pod warunkiem B określamy jako

$$\mathbf{P}(A|B) := \frac{\mathbf{P}(A \cap B)}{\mathbf{P}(B)}.$$

2.2. Pojęcia związane z teorią grafów[2]

Definicja 2.4 (graf)**.** *Grafem* nazywamy parę zbiorów $\Gamma = (U, E)$, gdzie U jest pewnym niepustym zbiorem, zaś $E \subseteq \{\{u, v\} \colon u, v \in U,\ u \neq v\}$ składa się z pewnej liczby dwuelementowych podzbiorów zbioru U. Elementy zbioru U nazywamy *wierzchołkami*, zaś elementy zbioru E *krawędziami* grafu Γ.

Jeśli $\{u, v\}$ jest krawędzią, to mówimy, że wierzchołki u i v są *końcami* tej krawędzi. Mówimy również, że wierzchołki u i v *sąsiadują* ze sobą: u jest *sąsiadem* v, a v jest sąsiadem u.

Definicja 2.5 (podgraf)**.** Jeżeli dane są grafy $\Gamma = (U, E)$, $\Gamma' = (U', E')$ oraz zachodzą związki $U' \subseteq U$, $E' \subseteq E$, to graf Γ' nazywamy *podgrafem* grafu Γ.

Definicja 2.6 (ścieżka)**.** Ciąg parami różnych wierzchołków $(v_0, v_1, \ldots, v_{k-1}, v_k)$ grafu $\Gamma = (U, E)$ taki, że $\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{k-2}, v_{k-1}\}, \{v_{k-1}, v_k\} \in E$, nazywamy *ścieżką* długości k. Wierzchołek v_0 nazywamy *początkiem*, zaś wierzchołek v_k *końcem* ścieżki. Mówimy też, że ścieżka *łączy* wierzchołki v_0 i v_k.

Innymi słowy, ścieżka to ciąg kolejno połączonych ze sobą różnych wierzchołków grafu.

Definicja 2.7 (cykl)**.** Ciąg wierzchołków $(v_0, v_1, \ldots, v_{k-1}, v_0)$ grafu $\Gamma = (U, E)$ taki, że $v_0, v_1, \ldots, v_{k-1}$ są parami różne oraz $\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{k-2}, v_{k-1}\}$, $\{v_{k-1}, v_0\} \in E$, nazywamy *cyklem* długości k.

Definicja 2.8 (graf spójny)**.** Graf $\Gamma = (U, E)$ nazywamy *spójnym*, jeżeli dla każdych dwóch wierzchołków $u, v \in U$ istnieje ścieżka, której początkiem jest u, a końcem v.

Innymi słowy, graf jest spójny, jeżeli każde dwa jego wierzchołki są połączone ścieżką.

Przykład 2.1 (graf, podgraf, ścieżka, cykl)**.** Na rysunku 2.1 przedstawiono przykładowy graf (cały rysunek) i jego podgraf (na czerwono), a także ścieżkę (na zielono) i cykl (na niebiesko) w tym grafie. Ten graf nie jest spójny, ponieważ wierzchołki u i v nie są połączone żadną ścieżką.

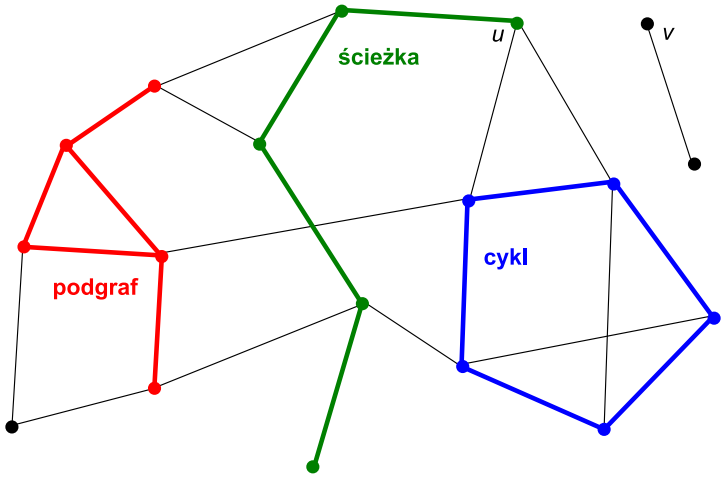
Definicja 2.9 (las)**.** *Lasem* nazywamy graf, który nie zawiera cykli.

Definicja 2.10 (drzewo)**.** *Drzewem* nazywamy spójny las.

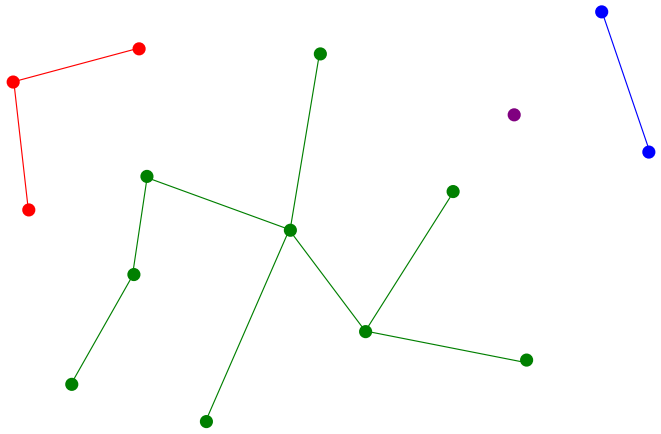
Przykład 2.2 (las, drzewo)**.** Na rysunku 2.2 przedstawiono przykładowy las. Każda z jego czterech składowych spójnych jest drzewem.

Definicja 2.11 (drzewo ukorzenione)**.** Jeżeli w drzewie wyróżnimy jeden z wierzchołków i nazwiemy go *korzeniem*, to takie drzewo nazywamy *drzewem ukorzenionym*, a jego wierzchołki *węzłami*.

[2] Na podstawie [14].



Rysunek 2.1. Przykład grafu zawierającego podgraf, ścieżkę i cykl.



Rysunek 2.2. Przykładowy las. Różne składowe spójności zaznaczono różnymi kolorami. Każda składowa spójności lasu jest drzewem.

Definicja 2.12 (przodek)**.** W drzewie ukorzenionym każdy węzeł na ścieżce łączącej korzeń z węzłem x nazywamy *przodkiem* węzła x.

Definicja 2.13 (potomek)**.** Każdy węzeł, którego przodkiem jest x, nazywamy *potomkiem* węzła x.

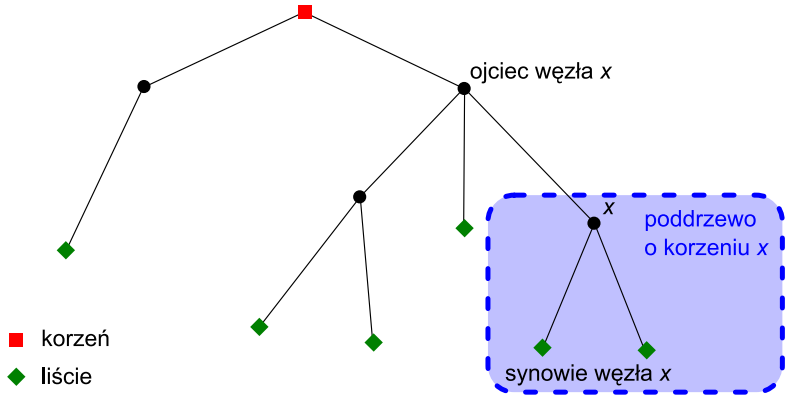
Definicja 2.14 (ojciec)**.** *Ojciec* węzła x to węzeł, który jest zarazem sąsiadem i przodkiem węzła x.

Definicja 2.15 (syn)**.** *Syn* węzła x to węzeł, który jest zarazem sąsiadem i potomkiem węzła x.

Definicja 2.16 (liść)**.** *Liść* to węzeł, który nie ma potomków.

Definicja 2.17 (poddrzewo)**.** Jeżeli x jest węzłem drzewa ukorzenionego Γ, to *poddrzewem* drzewa Γ o korzeniu x (lub zaczepionym w x) nazywamy podgraf drzewa Γ składający się ze wszystkich potomków węzła x i krawędzi je łączących.

Przykład 2.3 (drzewo ukorzenione)**.** Na rysunku 2.3 przedstawiono przykładowe drzewo z korzeniem (czerwony kwadrat) i zaznaczono jego liście (zielone



Rysunek 2.3. Przykładowe drzewo z korzeniem.

romby). Pokazano też ojca i synów wyróżnionego węzła x oraz poddrzewo zaczepione w tym węźle (w niebieskiej ramce).

Definicja 2.18 (graf z wagami)**.** *Grafem z wagami* nazywamy strukturę $\Gamma = (U, E, W)$, gdzie (U, E) jest grafem, zaś $W\colon E \to \mathbb{R}$ jest funkcją, która przyporządkowuje każdej krawędzi $e \in E$ wagę $W(e)$.

Definicja 2.19 (graf skierowany)**.** *Grafem skierowanym* nazywamy parę zbiorów $\Gamma = (U, E)$, gdzie U jest pewnym niepustym zbiorem, zaś $E \subseteq \{(u, v)\colon u, v \in U,\ u \neq v\} = U \times U$ jest zbiorem (uporządkowanych) par elementów zbioru U (czyli relacją dwuargumentową w U). Podobnie jak w przypadku grafu nieskierowanego, elementy zbioru U nazywamy *wierzchołkami*, zaś elementy zbioru E *krawędziami* grafu Γ.

Jeżeli $e = (u, v) \in E$ jest krawędzią grafu skierowanego $\Gamma = (U, E)$, to wierzchołek u nazywamy *początkiem*, a wierzchołek v *końcem* krawędzi e.

O grafie skierowanym można również myśleć jako o zwykłym grafie, którego każdej krawędzi przyporządkowano zwrot — inaczej mówiąc: nadano orientację.

Każde drzewo można przedstawić w postaci grafu skierowanego, nadając każdej jego krawędzi orientację od ojca do syna.

Definicja 2.20 (drzewo uporządkowane)**.** *Drzewem uporządkowanym* nazywamy graf skierowany powstały z drzewa ukorzenionego przez nadanie każdej krawędzi orientacji od ojca do syna, w którym synowie każdego węzła wewnętrznego zostali uporządkowani (od lewej do prawej).

Na zbiorze liści drzewa uporządkowanego można wprowadzić naturalny porządek w następujący sposób: Jeżeli liście x i y są synami tego samego węzła z, to ich wzajemne uporządkowanie pokrywa się z ich uporządkowaniem jako synów węzła z. Jeżeli liście x i y mają różnych ojców, to istnieje węzeł z, który jest potomkiem wszystkich wspólnych przodków liści x i y. Niech x' i y' będą takimi synami węzła z, że x' jest przodkiem liścia x, a y' jest przodkiem liścia y. Wówczas x poprzedza y (jako liść) wtedy i tylko wtedy, gdy x' poprzedza y' (jako syna węzła z), zaś y poprzedza x wtedy i tylko wtedy, gdy y' poprzedza x'.

Definicja 2.21 (multigraf skierowany)**.** *Multigrafem skierowanym* nazywamy strukturę $\Gamma = (U, E, \gamma)$, w której:

— U jest dowolnym niepustym zbiorem *wierzchołków*,

— E jest dowolnym zbiorem *krawędzi*,

— γ jest funkcją ze zbioru E do zbioru $U \times U$.

Jeżeli $e \in E$ jest krawędzią multigrafu skierowanego $\Gamma = (U, E)$ oraz $\gamma(e) = (u, v) \in U \times U$, to wierzchołek u nazywamy *początkiem*, a wierzchołek v *końcem* krawędzi e, mówimy też, że krawędź e *biegnie od* u *do* v lub że u z v jest połączone krawędzią.

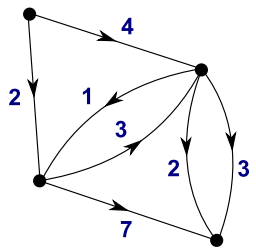
Definicja 2.22 (multigraf skierowany z wagami)**.** *Multigrafem skierowanym z wagami* nazywamy strukturę $\Gamma = (U, E, \gamma, W)$, gdzie (U, E, γ) jest multigrafem skierowanym, zaś $W\colon E \to \mathbb{R}$ jest funkcją, która przyporządkowuje każdej krawędzi $e \in E$ wagę $W(e)$.

Definicja 2.23 (ścieżka w multigrafie skierowanym)**.** Ciąg parami różnych wierzchołków $(v_0, v_1, \ldots, v_{k-1}, v_k)$ multigrafu skierowanego (multigrafu skierowanego z wagami) Γ taki, że wierzchołki v_0 z v_1, v_1 z v_2, …, v_{k-2} z v_{k-1}, v_{k-1} z v_k są kolejno połączone krawędziami e_1, e_2, … e_k, nazywamy *ścieżką*. Wierzchołek v_0 nazywamy *początkiem*, zaś wierzchołek v_k *końcem* ścieżki. Mówimy też, że ścieżka *łączy* wierzchołki v_0 i v_k. *Długością* tej ścieżki nazywamy liczbę $W(e_1) + W(e_2) + \cdots + W(e_k)$, gdzie $W(e)$ oznacza wagę krawędzi e.

Definicja 2.24 (odległość wierzchołków)**.** Niech dany będzie graf (graf skierowany, multigraf skierowany) z wagami oraz jego dowolne dwa wierzchołki u i v. *Odległością* z wierzchołka u do wierzchołka v nazywamy długość najkrótszej ścieżki łączącej wierzchołek u z wierzchołkiem v i oznaczamy ją przez $d(u, v)$.

Definicja 2.25 (cykl w multigrafie skierowanym)**.** Ciąg wierzchołków $(v_0, v_1, \ldots, v_{k-1}, v_0)$ multigrafu skierowanego (multigrafu skierowanego z wagami) Γ taki, że wierzchołki $v_0, v_1, \ldots, v_{k-1}$ są parami różne, a wierzchołki v_0 z v_1, v_1 z v_2, …, v_{k-2} z v_{k-1}, v_{k-1} z v_0 są kolejno połączone krawędziami, nazywamy *cyklem* długości k.

Definicja 2.26 (acykliczny multigraf skierowany)**.** Multigraf skierowany (multigraf skierowany z wagami) nazywamy *acyklicznym*, jeżeli nie zawiera cykli.



Rysunek 2.4. Przykładowy multigraf skierowany z wagami. Niebieskie liczby to wagi krawędzi.

Przykład 2.4 (multigraf skierowany z wagami)**.** Na rysunku 2.4 przedstawiono przykładowy multigraf skierowany z wagami. Multigraf ten nie ma cykli, jest zatem acyklicznym multigrafem skierowanym.

2.3. Gramatyki i języki formalne[3]

2.3.1. Symbole, alfabety i łańcuchy

Pojęcie 2.27 (symbol)**.** *Symbol* jest pojęciem pierwotnym — nie definiuje się go.

Symbolem może być pojedynczy znak — litera, cyfra. W przypadku przetwarzania języka naturalnego symbolami są najczęściej poszczególne wyrazy.

Definicja 2.28 (alfabet, słownik)**.** *Alfabetem* albo *słownikiem* nazywamy skończony zbiór symboli.

Na ogół, zwłaszcza jeśli symbolami są litery, używa się określenia *alfabet*. Jeżeli symbolami są wyrazy, wygodniejsze i bardziej naturalne jest używanie pojęcia *słownik*.

Przykład 2.5 (alfabety, słowniki)**.** Alfabetami (słownikami) mogą być na przykład następujące zbiory:
— $\{0, 1\}$,
— $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$,
— $\{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$,
— $\{A, Ą, B, C, Ć, D, E, Ę, F, G, H, I, J, K, L, Ł, M, N, Ń, O, Ó, P, R, S, Ś, T, U, W, Y, Z, Ź, Ż\}$,
— $\{A, \varrho, q, \Psi, \omega\}$,
— $\{S\,,\ NP\,,\ N\,,\ VP\,,\ V\,\}$,
— $\{\mathit{człowiek}\,,\ \mathit{kot}\,,\ \mathit{pies}\,,\ \mathit{mówi}\,,\ \mathit{szczeka}\,\}$.

Definicja 2.29 (łańcuch)**.** Skończony ciąg zestawionych obok siebie symboli alfabetu V nazywamy *łańcuchem nad alfabetem* V lub po prostu *łańcuchem*.

Łańcuch nazywany bywa również *słowem*, ale w tej pracy będę raczej unikał tego określenia, ponieważ w przykładach symbolami najczęściej będą wyrazy, zaś łańcuchami — ciągi wyrazów. Nazywanie ciągu wyrazów słowem mogłoby prowadzić do nieporozumień.

Uwaga 2.1. W przypadku gdy symbolami będą wyrazy, dla przejrzystości zapisu będę oddzielał je spacjami. Tych spacji nie należy traktować jako oddzielnych symboli, lecz jedynie jako konwencję notacyjną.

Przykład 2.6 (łańcuchy)**.** Niech $V = \{a, b, d, k, r\}$. Wówczas *abrakadabra*, *barak* i *dakar* są łańcuchami nad alfabetem V.

Przykład 2.7 (łańcuch)**.** *być albo nie być* jest łańcuchem nad słownikiem $\{\mathit{albo}\,,\ \mathit{być}\,,\ \mathit{i}\,,\ \mathit{mieć}\,,\ \mathit{nie}\,,\ \mathit{tak}\,\}$.

Definicja 2.30 (długość łańcucha)**.** *Długość* łańcucha to liczba tworzących go symboli. Długość łańcucha w oznaczamy przez $|w|$.

Przykład 2.8 (długość łańcucha)**.** Łańcuch *Ala ma kota* ma długość 3, ponieważ składa się z 3 symboli: $\mathit{Ala}\,,\ \mathit{ma}$ i $\mathit{kota}\,.$

Definicja 2.31 (łańcuch pusty)**.** Łańcuch o długości 0 (składający się z zera symboli) nazywa się *łańcuchem pustym* i oznacza symbolem ϵ.

Łańcuch pusty jest łańcuchem nad każdym alfabetem.

[3] Na podstawie [6].

Definicja 2.32 (konkatenacja)**.** *Konkatenacją (złożeniem)* dwóch łańcuchów nazywamy łańcuch powstały przez wypisanie kolejno wszystkich symboli pierwszego łańcucha, a zaraz po nich, bez żadnej przerwy, kolejno wszystkich symboli drugiego łańcucha. Operację konkatenacji oznaczamy przez zestawienie bezpośrednio obok siebie oznaczeń składanych łańcuchów.

Przykład 2.9 (konkatenacja)**.** Konkatenacją łańcucha *wlazł kotek* i łańcucha *na płotek* jest łańcuch *wlazł kotek na płotek*.

Przykład 2.10 (konkatenacja)**.** Jeżeli symbolami są poszczególne litery, to konkatenacją łańcuchów *oko* i *liczność* jest łańcuch *okoliczność*.

Przykład 2.11 (konkatenacja — oznaczenie)**.** Konkatenację łańcuchów oznaczonych przez x i y zapisujemy jako xy.

Operacja konkatenacji jest łączna, możemy zatem mówić o konkatenacji więcej niż dwóch łańcuchów.

Przykład 2.12 (konkatenacja kilku łańcuchów)**.** Konkatenacją łańcuchów w, x, y i z jest łańcuch $wxyz = ((wx)y)z$.

Definicja 2.33 (podłańcuch)**.** Jeżeli u, v, w, w' są łańcuchami oraz $w = uw'v$, to mówimy, że w' jest *podłańcuchem* łańcucha w.

Przykład 2.13 (podłańcuchy)**.** Łańcuchy ϵ, *pchły koło*, *szły*, *koło wody* i *szły pchły koło wody* są podłańcuchami łańcucha *szły pchły koło wody*.

2.3.2. Języki

Definicja 2.34 (język)**.** *Język* jest zbiorem łańcuchów nad pewnym alfabetem.

Przykład 2.14 (języki)**.** Niech $V = \{0, 1, 2\}$. Wówczas językami złożonymi z łańcuchów nad alfabetem V są m.in.:
- $\emptyset$,
- $\{\epsilon\}$,
- $\{100, 101, 102\}$,
- $\{\epsilon, 0, 00, 1, 11, 111\}$,
- $\{2, 22, 222, 2222, 22222, \ldots\}$,
- $\{\epsilon, 0, 1, 2, 00, 01, 02, 10, 11, 12, 20, 21, 22, 000, \ldots, 222, 0000, \ldots, 2222, \ldots\}$.

Jak widać z powyższych przykładów, języki mogą być zarówno zbiorami skończonymi, jak i nieskończonymi. Ostatni przykład, zawierający wszystkie możliwe łańcuchy nad danym alfabetem, jest przypadkiem szczególnym:

Definicja 2.35. Niech V będzie alfabetem. Wówczas język złożony ze wszystkich łańcuchów nad alfabetem V oznaczamy przez V^*.

Przykład 2.15. $\{1, 2\}^* = \{\epsilon, 1, 2, 11, 12, 21, 22, 111, \ldots, 222, 1111, \ldots, 2222, \ldots\}$.

Definicja 2.36. Język złożony ze wszystkich niepustych łańcuchów nad alfabetem V oznaczamy przez V^+.

Przykład 2.16. $\{\, ha \,\}^+ = \{\, ha\,,\ ha\ ha\,,\ ha\ ha\ ha\,,\ ha\ ha\ ha\ ha\,, \ldots\}$.

Zachodzą następujące związki: $V^+ = V^* \setminus \{\epsilon\}$, $V^* = V^+ \cup \{\epsilon\}$.

2.3.3. Gramatyki formalne

Definicja 2.37 (gramatyka)**.** *Gramatyką (kombinatoryczną)* nazywamy strukturę $G = (V, T, R, S)$, gdzie:
- V jest alfabetem, nazywanym *alfabetem symboli pomocniczych (nieterminalnych)* albo *zmiennych*,
- T jest rozłącznym z V alfabetem, nazywanym *alfabetem symboli końcowych (terminalnych)*,
- R jest zbiorem *produkcji (reguł)*, czyli napisów postaci $\zeta \to \omega$, gdzie $\zeta \in (V \cup T)^+$, $\omega \in (V \cup T)^*$,
- symbol $S \in V$ nazywany jest *symbolem początkowym*.

Jeżeli $\zeta \to \omega$ jest regułą produkcji, to łańcuch ζ nazywamy jej *poprzednikiem*, a łańcuch ω — *następnikiem*.

Ze względu na postać napisów produkcji można wśród gramatyk formalnych wyróżnić pewne podkategorie.

Definicja 2.38 (gramatyka kontekstowa)**.** Jeżeli każda produkcja ma postać $\zeta A \xi \to \zeta \omega \xi$, $A \in V$, $\zeta, \xi \in (V \cup T)^*$, $\omega \in (V \cup T)^+$, to taką gramatykę nazywamy *gramatyką kontekstową*.

Łańcuchy u i v z powyższej definicji często nazywa się *kontekstem* zmiennej A.

Definicja 2.39 (gramatyka bezkontekstowa z ϵ-regułami)**.** *Gramatyką bezkontekstową z ϵ-regułami* nazywamy strukturę $G = (V, T, R, S)$ składającą się z:
- alfabetu symboli pomocniczych V,
- alfabetu symboli końcowych T,
- zbioru R produkcji postaci $A \to \omega$, $A \in V$, $\omega \in (V \cup T)^*$,
- symbolu początkowego $S \in V$.

Definicja 2.40 (gramatyka bezkontekstowa)**.** *Gramatyką bezkontekstową (ϵ-wolną)* nazywamy strukturę $G = (V, T, R, S)$ składającą się z:
- alfabetu symboli pomocniczych V,
- alfabetu symboli końcowych T,
- zbioru R produkcji postaci $A \to \omega$, $A \in V$, $\omega \in (V \cup T)^+$.
- symbolu początkowego $S \in V$,

Definicja 2.41 (gramatyka monotoniczna)**.** Jeżeli $G = (V, T, R, S)$ jest taką gramatyką, że każda produkcja z R jest postaci $\zeta \to \xi$, gdzie $|\zeta| \leqslant |\xi|$, to G nazywamy *gramatyką monotoniczną*.

Twierdzenie 2.3. *Gramatyka bezkontekstowa ϵ-wolna jest monotoniczna.*

Dowód. Teza wynika bezpośrednio z faktu, że gramatyka bezkontekstowa ϵ-wolna nie zawiera reguł ze słowem pustym po prawej stronie, dlatego $|A| = 1 \leqslant |\omega|$ dla każdej produkcji $A \to \omega \in R$. □

Przykład 2.17 (prosta gramatyka bezkontekstowa)**.** Prostym przykładem gramatyki bezkontekstowej może być struktura $G = (V, T, R, S)$:
- $V = \{S, A\}$,
- $T = \{1\}$,
- $R = \{S \to A1,\ S \to 1A,\ A \to 11\}$,
- $S \in V$ jest symbolem początkowym.

Przykład 2.18 (gramatyka bezkontekstowa — „śmiech”)**.** Przykładem gramatyki bezkontekstowej jest gramatyka $G = (V, T, R, S)$, która (jak się przekonamy w przykładzie 2.21) generuje napisy przypominające śmiech:
- $V = \{ S \}$,
- $T = \{ ha \}$,
- $R = \{ S \to S\ S,\ S \to ha \}$,
- $S \in V$ jest symbolem początkowym.

Przykład 2.19 (gramatyka bezkontekstowa)**.** Innym przykładem gramatyki bezkontekstowej jest następująca struktura $G = (V, T, R, S)$:
- $V = \{ S, VP, V, NPN, NPG, NPA, NN, NG, NA, PP \}$,
- $T = \{ butów, buty, chowa, do, pasta, pastę, pasty, szewc, szewca \}$,
- $R = \{ S \to NPN\ VP,\ VP \to V\ NPA,\ VP \to V\ NPA\ PP,$
  $V \to chowa,\ PP \to do\ NPG,\ NPN \to NN,\ NPN \to NN\ PP,$
  $NPG \to NG,\ NPG \to NG\ PP,\ NPA \to NA,\ NPA \to NA\ PP,$
  $NN \to szewc,\ NN \to pasta,\ NN \to buty,$
  $NG \to szewca,\ NG \to pasty,\ NG \to butów,$
  $NA \to szewca,\ NA \to pastę,\ NA \to buty \}$,
- $S \in V$ jest symbolem początkowym.

Definicja 2.42 (zbinaryzowana gramatyka bezkontekstowa)**.** Gramatykę bezkontekstową $G = (V, T, R, S)$ nazywamy *zbinaryzowaną*, jeżeli każda jej produkcja jest postaci $A \to X$ lub $A \to XY$, gdzie $A \in V$, $X, Y \in V \cup T$.

Definicja 2.43 (postać normalna Chomsky’ego)**.** Mówimy, że gramatyka bezkontekstowa $G = (V, T, R, S)$ jest w *postaci normalnej Chomsky’ego*, jeżeli każda jej produkcja jest postaci $A \to BC$ lub $A \to a$, gdzie $A, B, C \in V$, $a \in T$.

Definicja 2.44 (bezpośrednia wyprowadzalność)**.** Niech $G = (V, T, R, S)$ będzie gramatyką oraz $\zeta, \xi, \omega, \omega' \in (V \cup T)^*$. Mówimy, że łańcuch $\zeta\omega'\xi$ jest *bezpośrednio wyprowadzalny* z łańcucha $\zeta\omega\xi$ w gramatyce G, jeśli istnieje produkcja $\omega \to \omega' \in R$. Piszemy wówczas

$$\zeta\omega\xi \Rightarrow_G \zeta\omega'\xi$$

albo (jeśli wiadomo, o którą gramatykę chodzi) po prostu

$$\zeta\omega\xi \Rightarrow \zeta\omega'\xi.$$

Definicja 2.45 (wyprowadzenie)**.** Niech $G = (V, T, R, S)$ będzie gramatyką oraz $\omega, \omega' \in (V \cup T)^*$. Ciąg łańcuchów $\zeta_1, \zeta_2, \ldots, \zeta_n \in (V \cup T)^*$ taki, że

$$\omega = \zeta_1 \Rightarrow_G \zeta_2 \Rightarrow_G \ldots \Rightarrow_G \zeta_n = \omega',$$

nazywamy *wyprowadzeniem* łańcucha ω' z łańcucha ω w gramatyce G.

Definicja 2.46 (wyprowadzalność)**.** Mówimy, że łańcuch ω' jest *wyprowadzalny* z łańcucha ω w gramatyce G, jeżeli istnieje wyprowadzenie łańcucha ω' z łańcucha ω w gramatyce G. Piszemy wówczas

$$\omega \Rightarrow_G^* \omega'$$

albo (jeśli wiadomo, o którą gramatykę chodzi) po prostu

$$\omega \Rightarrow^* \omega'.$$

W szczególności dla każdego symbolu X gramatyki G zachodzi związek

$$X \Rightarrow_G^* X.$$

Przykład 2.20 (wyprowadzenie)**.** W gramatyce z przykładu 2.19 łańcuch *pasta do butów* jest wyprowadzalny z symbolu NPN, ponieważ:

$$NPN \overset{①}{\Rightarrow} NN\ PP \overset{②}{\Rightarrow} NN\ do\ NPG \overset{③}{\Rightarrow} \Rightarrow pasta\ do\ NPG \overset{④}{\Rightarrow} pasta\ do\ NG \overset{⑤}{\Rightarrow} pasta\ do\ butów\,,$$

① $NPN \to NN\ PP \in R$,
② $PP \to do\ NPG \in R$,
③ $NN \to pasta \in R$,
④ $NPG \to NG \in R$,
⑤ $NG \to butów \in R$.

Możemy zatem napisać:

$$NPN \Rightarrow^* pasta\ do\ butów\,.$$

Definicja 2.47 (język generowany przez gramatykę)**.** Niech $G = (V, T, R, S)$ będzie gramatyką. Wówczas zbiór łańcuchów

$$L(G) := \{w \in T^* \colon S \Rightarrow_G^* w\}$$

nazywamy *językiem generowanym przez gramatykę* G.

Innymi słowy, język generowany przez daną gramatykę to zbiór łańcuchów symboli końcowych, które można wyprowadzić z jej symbolu początkowego przy użyciu jej reguł produkcji.

Jeżeli dwie gramatyki generują ten sam język, to mówimy, że są one *równoważne*.

Definicja 2.48 (język bezkontekstowy)**.** Każdy język taki, że istnieje gramatyka bezkontekstowa, która go generuje, nazywany jest *językiem bezkontekstowym*.

Przykład 2.21 (język generowany przez gramatykę — „śmiech")**.** Językiem generowanym przez gramatykę G z przykładu 2.18 jest zbiór napisów przypominających śmiech:

$$L(G) = \{\, ha\,,\ ha\ ha\,,\ ha\ ha\ ha\,, \ldots\}.$$

Przykład ten pokazuje w szczególności, że język $\{\, ha\,,\ ha\ ha\,,\ ha\ ha\ ha\,, \ldots\}$ jest językiem bezkontekstowym.

Przykład 2.22 (język generowany przez gramatykę)**.** Językiem generowanym przez gramatykę G z przykładu 2.17 jest zbiór jednoelementowy:

$$L(G) = \{111\}.$$

Jest to język bezkontekstowy.

Twierdzenie 2.4. *Każdą gramatykę bezkontekstową można zbinaryzować: dla każdej gramatyki bezkontekstowej G istnieje zbinaryzowana gramatyka bezkontekstowa G' generująca ten sam język, tj. taka, że $L(G') = L(G)$.*

Dowód.[4] Niech dana będzie gramatyka bezkontekstowa $G = (V, T, R, S)$. Niech

$$R_{>2} := \{A \to X_1 \dots X_k \in R \colon k > 2, A \in V, X_1, \dots, X_k \in V \cup T\}$$

oznacza zbiór tych reguł, które mają po prawej stronie więcej niż dwa symbole. Dla każdej reguły $r = A_r \to X_{r,1} \dots X_{r,k} \in R_{>2}$ tworzymy nowe symbole pomocnicze:

$$V_r := \{C_{r,3}, \dots, C_{r,k-1}, C_{r,k}\},$$

oraz reguły:

$$R_r := \{A_r \to C_{r,k}X_{r,k},\ C_{r,k} \to C_{r,k-1}X_{r,k-1},\ \dots \\ \dots,\ C_{r,4} \to C_{r,3}X_{r,3},\ C_{r,3} \to X_{r,1}X_{r,2}\}.$$

Gramatyka $G' = (V', T, R', S)$, w której

$$V' := V \cup \bigcup_{r \in R_{>2}} V_r,$$
$$R' := (R \setminus R_{>2}) \cup \bigcup_{r \in R_{>2}} R_r,$$

jest wówczas zbinaryzowaną gramatyką bezkontekstową i generuje język $L(G') = L(G)$.

Istotnie, gramatyka G' jest zbinaryzowana, ponieważ nie zawiera produkcji, które miałyby po prawej stronie więcej niż dwa symbole.

Pokażemy teraz równość generowanych języków.

Niech $w \in L(G)$. Oznacza to, że łańcuch w posiada wyprowadzenie z symbolu początkowego S w gramatyce G:

$$S \ = \ \omega_0 \ \Rightarrow_G \ \omega_1 \ \Rightarrow_G \ \dots \ \Rightarrow_G \ \omega_m \ = \ w.$$

Jeżeli $\omega_{i-1} \Rightarrow_G \omega_i$ przy użyciu reguły, która ma co najwyżej dwa symbole po prawej stronie, to także

$$\omega_{i-1} \ \Rightarrow_{G'} \ \omega_i.$$

Jeżeli zaś $\omega_{i-1} \Rightarrow_G \omega_i$ za pomocą reguły $r = A_r \to X_{r,1} \dots X_{r,k} \in R$, $k > 2$, czyli

$$\omega_{i-1} \ = \ \zeta A_r \xi \ \Rightarrow_G \ \zeta X_{r,1} \dots X_{r,k} \xi \ = \ \omega_i,$$

to

$$\omega_{i-1} \ = \ \zeta A_r \xi \ \Rightarrow_{G'} \ \zeta C_{r,k} X_{r,k} \xi \ \Rightarrow_{G'} \ \zeta C_{r,k-1} X_{r,k-1} X_{r,k} \xi \ \Rightarrow_{G'} \ \dots \\ \dots \ \Rightarrow_{G'} \ \zeta C_{r,3} X_{r,3} \dots X_{r,k} \xi \ \Rightarrow_{G'} \ \zeta X_{r,1} X_{r,2} X_{r,3} \dots X_{r,k} \xi \ = \ \omega_i,$$

czyli

$$\omega_{i-1} \ \Rightarrow^*_{G'} \ \omega_i.$$

Wynika stąd, że

$$S \ \Rightarrow^*_{G'} \ w,$$

a zatem

$$w \in L(G').$$

Ostatecznie:

$$L(G) \subseteq L(G').$$

[4] Ten dowód w oparciu o dowód twierdzenia 2.5 według [6].

Pokażemy teraz, że jeżeli $A \Rightarrow_{G'}^{*} w$, to $A \Rightarrow_{G}^{*} w$ dla $A \in V$, $w \in T^{+}$. Niech

$$A = \omega_0 \Rightarrow_{G'} \omega_1 \Rightarrow_{G'} \ldots \Rightarrow_{G'} \omega_m = w$$

będzie wyprowadzeniem łańcucha w z symbolu A w gramatyce G'. Pokażemy przez indukcję względem długości m tego wyprowadzenia, że łańcuch w posiada również wyprowadzenie z symbolu A w gramatyce G'.

Dla $m = 1$ teza jest oczywista.

Załóżmy teraz, że każdy łańcuch symboli terminalnych posiadający wyprowadzenie o długości nie większej niż m w gramatyce G' z pewnego symbolu $A \in V$ posiada również wyprowadzenie z symbolu A w gramatyce G. Niech

$$A = \omega_0 \Rightarrow_{G'} \omega_1 \Rightarrow_{G'} \ldots \Rightarrow_{G'} \omega_m \Rightarrow_{G'} \omega_{m+1} = w$$

będzie wyprowadzeniem łańcucha w z symbolu A w gramatyce G'. Rozważmy produkcję r', która wyprowadza bezpośrednio łańcuch ω_1 z symbolu A. Ponieważ $r' \in R'$, to po prawej stronie tej produkcji mogą być co najwyżej dwa symbole.

Jeżeli wszystkie symbole po prawej stronie produkcji r' należą do zbioru V, to:

— $r' \in R$,

— $\omega_1 \in V^{+}$ i na mocy założenia indukcyjnego istnieje wyprowadzenie łańcucha w z łańcucha ω_1 w gramatyce G,

zatem istnieje wyprowadzenie łańcucha w z symbolu A w gramatyce G.

Jeżeli zaś któryś z symboli po prawej stronie rozważanej produkcji r' jest spoza zbioru V, to produkcja ta musi być postaci

$$r' = A \rightarrow CX,$$

gdzie $A \in V$, $C \in V' \setminus V$, $X \in V \cup T$. Ale jeżeli $C \in V' \setminus V$, to znaczy, że istnieje reguła $r = A \rightarrow X_{r,1} \ldots X_{r,k} \in R$ taka, że $C = C_{r,k} \in V_r$. Ponieważ wszystkie symbole nieterminalne występujące w wyprowadzeniu łańcucha symboli terminalnych muszą zostać rozwinięte, oznacza to, że w rozważanym wyprowadzeniu łańcucha w musiały zostać użyte wszystkie reguły ze zbioru R_r. Łańcuch w możemy więc zapisać w postaci

$$w = w_{(1)} \ldots w_{(k-1)} w_{(k)},$$

gdzie $X_{r,i} \Rightarrow_{G'}^{*} w_{(i)} \in T^{+}$ dla $i = 1, \ldots, k$. Na mocy założenia indukcyjnego dla każdego $i = 1, \ldots, k$ zachodzi

$$X_{r,i} \Rightarrow_{G}^{*} w_{(i)}.$$

Oznacza to, że

$$A \Rightarrow_{G} X_{r_1} \ldots X_{r_k} \Rightarrow^{*} w_{(1)} \ldots w_{(k)} = w,$$

a zatem

$$A \Rightarrow_{G}^{*} w,$$

co dowodzi kroku indukcyjnego, więc również i całej implikacji.

Pokazaliśmy, że jeżeli $A \Rightarrow_{G'}^{*} w$, to $A \Rightarrow_{G}^{*} w$ dla $A \in V$, $w \in T^{+}$. Wynika stąd w szczególności, że jeśli $S \Rightarrow_{G'}^{*} w$, to $S \Rightarrow_{G}^{*} w$ dla $w \in T^{+}$, czyli że $L(G') \subseteq L(G)$.

Ponieważ pokazaliśmy wcześniej, że $L(G) \subseteq L(G')$, otrzymujemy równość generowanych języków:

$$L(G') = L(G).$$

□

Twierdzenie 2.5. *Każdy język bezkontekstowy jest generowany przez pewną gramatykę bezkontekstową w postaci normalnej Chomsky'ego.*

Dowód. Dowód, polegający na zastąpieniu produkcji niebędących w postaci normalnej Chomsky'ego równoważnym zestawem produkcji postaci $A \to BC$ i $A \to a$, przedstawiony jest między innymi w [6]. Używa się w nim metod podobnych do tych zastosowanych w dowodzie twierdzenia 2.4. □

Definicja 2.49 (wyprowadzenie lewostronne)**.** Niech $G = (V, T, R, S)$ będzie gramatyką bezkontekstową. Wyprowadzenie

$$A = \zeta_0 \Rightarrow_G \zeta_1 \Rightarrow_G \ldots \Rightarrow_G \zeta_n = w,$$

nazywamy *wyprowadzeniem lewostronnym* łańcucha symboli terminalnych w z symbolu pomocniczego A, jeżeli w każdym kolejnym kroku wyprowadzenia stosujemy produkcję do symbolu pomocniczego leżącego najbardziej na lewo. Innymi słowy, żeby nazwać wyprowadzenie lewostronnym, dla każdego $i = 1, \ldots, n$ muszą zachodzić warunki:

— $\zeta_{i-1} = uB\xi$,
— $\zeta_i = u\omega\xi$,
— istnieje reguła $B \to \omega \in R$,

dla pewnych $u \in T^*$, $B \in V$, $\xi, \omega \in (V \cup T)^*$.

Na ogół będziemy przyjmować, że chodzi o wyprowadzenie lewostronne z symbolu początkowego, chyba że będzie wyraźnie zaznaczone inaczej.

Uwaga 2.2. Dany łańcuch języka może mieć kilka różnych wyprowadzeń lewostronnych.

Przykład 2.23 (różne wyprowadzenia lewostronne jednego łańcucha)**.** Jedyny łańcuch języka generowanego przez gramatykę z przykładu 2.17 ma dwa różne wyprowadzenia lewostronne:

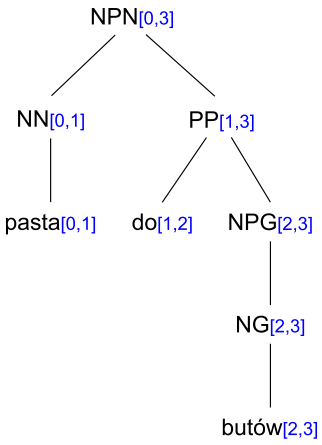
$$S \Rightarrow A1 \Rightarrow 111$$

oraz

$$S \Rightarrow 1A \Rightarrow 111.$$

Definicja 2.50 (drzewo wyprowadzenia)**.** Niech $G = (V, T, R, S)$ będzie gramatyką bezkontekstową. *Drzewem wyprowadzenia* albo *drzewem rozkładu* łańcucha $w \in L(G)$ nazywamy drzewo uporządkowane Γ spełniające następujące warunki:

(1) każdy węzeł drzewa Γ posiada etykietę będącą symbolem ze zbioru $V \cup T$,
(2) korzeń drzewa posiada etykietę S,
(3) każdy węzeł wewnętrzny drzewa Γ jest etykietowany symbolem pośrednim,
(4) jeśli wierzchołek opatrzony etykietą $A \in V$ ma synów opatrzonych etykietami kolejno $X_1, \ldots, X_k \in V \cup T$, to istnieje produkcja $A \to X_1 \ldots X_k \in R$.



Rysunek 2.5. Drzewo wyprowadzenia łańcucha *pasta do butów* z symbolu *NPN*.

Ponieważ w drzewie wyprowadzenia zdarza się niejednokrotnie, że różne wierzchołki etykietowane są tymi samymi symbolami gramatyki, a będziemy często potrzebować odróżniać takie wierzchołki, wprowadzamy dodatkową notację. Niech $G = (V, T, R, S)$ będzie gramatyką bezkontekstową, zaś $w = w_1 \dots w_n \in L(G)$. Jeżeli liśćmi poddrzewa o korzeniu w wierzchołku x o etykiecie $A \in V$ są kolejno $w_{i+1} \dots w_j$, to do etykiety A dołączamy parę liczb $[i, j]$ i nazywamy ją *zakresem* wierzchołka x. W skrócie będziemy pisać, że etykietą wierzchołka x jest $A[i, j]$.

Twierdzenie 2.6 (równoważność wyprowadzeń lewostronnych i drzew rozkładu)**.** *Każdemu wyprowadzeniu lewostronnemu (łańcucha symboli terminalnych) odpowiada dokładnie jedno drzewo rozkładu, a każdemu drzewu rozkładu odpowiada dokładnie jedno wyprowadzenie lewostronne. Korzeń drzewa jest etykietowany symbolem źródłowym wyprowadzenia, zaś etykiety liści drzewa odczytywane kolejno tworzą łańcuch wynikowy wyprowadzenia.*

Dowód. Dowód tego twierdzenia można znaleźć w [6]. □

Przykład 2.24 (drzewo wyprowadzenia)**.** Drzewo wyprowadzenia łańcucha *pasta do butów* z symbolu *NPN* z przykładu 2.20 w gramatyce z przykładu 2.19 jest pokazane na rysunku 2.5. Drzewu temu odpowiada wyprowadzenie lewostronne

$$\begin{aligned} NPN \Rightarrow NN\ PP \Rightarrow pasta\ PP \Rightarrow \\ \Rightarrow pasta\ do\ NPG \Rightarrow pasta\ do\ NG \Rightarrow pasta\ do\ butów\,. \end{aligned}$$

2.3.4. Probabilistyczne gramatyki bezkontekstowe[5]

Definicja 2.51 (gramatyka bezkontekstowa z wagami)**.** Piątkę uporządkowaną $G = (V, T, R, S, P)$ nazywamy *gramatyką bezkontekstową z wagami*, jeżeli (V, T, R, S) jest gramatyką bezkontekstową, zaś P jest funkcją z R do $\mathbb{R}$. Wartość $P(r)$ dla $r \in R$ nazywamy wówczas *wagą* reguły r.

Szczególnym rodzajem gramatyki bezkontekstowej z wagami jest probabilistyczna gramatyka bezkontekstowa.

[5] Na podstawie [10].

Definicja 2.52 (probabilistyczna gramatyka bezkontekstowa)**.** *Probabilistyczną (stochastyczną) gramatyką bezkontekstową* nazywamy piątkę uporządkowaną $G = (V, T, R, S, P)$, gdzie:

— V jest alfabetem symboli pomocniczych,
— T jest alfabetem symboli końcowych,
— R jest zbiorem produkcji postaci $A \to \omega$, $A \in V$, $\omega \in (V \cup T)^+$,
— $S \in V$ jest symbolem początkowym,
— $P\colon R \to [0, 1]$ jest funkcją taką, że

$$\bigvee_{A \in V} \sum_{(A \to \omega) \in R} P(A \to \omega) \;=\; 1.$$

Innymi słowy, probabilistyczna gramatyka bezkontekstowa powstaje z gramatyki bezkontekstowej przez dołączenie do każdej reguły pewnej wartości z przedziału $[0, 1]$ w taki sposób, żeby wartości przyporządkowane produkcjom o tym samym poprzedniku sumowały się do 1.

Każda probabilistyczna gramatyka bezkontekstowa jest gramatyką bezkontekstową z wagami.

Definicje postaci normalnej Chomsky'ego, wyprowadzeń, wyprowadzalności, języka generowanego, drzewa wyprowadzenia czy zakresu dla probabilistycznych gramatyk bezkontekstowych niczym nie różnią się od analogicznych definicji dla gramatyk nieprobabilistycznych. Podobnie rzecz ma się, jeśli chodzi o twierdzenia i własności. Wystarczy zauważyć, że dla każdej probabilistycznej gramatyki bezkontekstowej (V, T, R, S, P) czwórka (V, T, R, S) jest (nieprobabilistyczną) gramatyką bezkontekstową.

W językach naturalnych niektóre łańcuchy i konstrukcje występują częściej, inne — rzadziej. Pojęcie gramatyk probabilistycznych stworzono, aby można było modelować i badać probabilistyczne własności języków generowanych przez gramatyki, na przykład określać, jakie jest prawdopodobieństwo otrzymania danego łańcucha z symbolu początkowego gramatyki. Z tego powodu wprowadza się funkcję P, której wartość na każdej regule ma odzwierciedlać prawdopodobieństwo zastosowania tej reguły w wyprowadzeniu.

Określone przy pomocy funkcji P prawdopodobieństwo łańcucha powinno spełniać następujące warunki:
— prawdopodobieństwo podłańcucha nie zależy od jego położenia w łańcuchu (*niezależność od położenia*),
— prawdopodobieństwo podłańcucha nie zależy od sąsiadujących z nim podłańcuchów (*niezależność od kontekstu*),
— prawdopodobieństwo poddrzewa drzewa wyprowadzenia łańcucha nie zależy od symboli, z których wyprowadzono korzeń tego drzewa (*niezależność od przodków*).

Można wyobrazić sobie proces wyprowadzania łańcucha z symbolu początkowego jako następującą procedurę:

1. Rozpoczynamy od symbolu początkowego i ustawiamy go jako bieżący symbol.
2. Ze zbioru reguł wybieramy jedną z produkcji, których poprzednikiem jest bieżący symbol.
3. Przekształcamy bieżący łańcuch zgodnie z wybraną regułą, otrzymując nowy łańcuch.
4. Wybieramy z nowego łańcucha jeden z symboli i ustawiamy jako bieżący.

5. Powtarzamy kroki od 2 do 4, dopóki nie otrzymamy łańcucha złożonego w całości z symboli końcowych.

Przy takiej interpretacji wartość funkcji $P(A \to \omega)$ wyraża prawdopodobieństwo, że jeżeli bieżącym symbolem jest A, to wybierzemy regułę $A \to \omega$ (spośród wszystkich reguł o poprzedniku A). Przestrzenią zdarzeń elementarnych (dla każdego symbolu A inną!) jest wówczas zbiór $\Omega_A = \{A \to \omega \in R\}$ wszystkich produkcji gramatyki G mających A za poprzednika. Z tego powodu funkcja P nazywana jest *prawdopodobieństwem reguły*.

Przyjmując, że A oznacza zarówno (bieżący) symbol, jak i zdarzenie, że symbol ten został wybrany jako bieżący w przedstawionej powyżej procedurze, można napisać, że

$$P(A \to \omega) = \mathbf{P}(A \to \omega | A).$$

Zapis taki przypomina o tym, że w istocie funkcja P jest prawdopodobieństwem warunkowym, ponieważ dla różnych poprzedników reguły jest określona na różnych przestrzeniach probabilistycznych.

W poniższych przykładach prawdopodoieństwa reguł zostały dobrane arbitralnie, nie odzwierciedlają rzeczywistych prawdopodobieństw zastosowania tych reguł w języku polskim.

Przykład 2.25 (probabilistyczna gramatyka bezkontekstowa)**.** Z gramatyki z przykładu 2.19 można uczynić probabilistyczną gramatykę bezkontekstową przyporządkowując jej regułom prawdopodobieństwa w odpowiedni sposób:

— $G = (V, T, R, S, P)$,
— $V = \{ S, VP, V, NPN, NPG, NPA, NN, NG, NA, PP \}$,
— $T = \{ \textit{butów}, \textit{buty}, \textit{do}, \textit{pasta}, \textit{pastę}, \textit{pasty}, \textit{chowa}, \textit{szewc}, \textit{szewca} \}$,
— $R = \{ S \to NPN\ VP,\ VP \to V\ NPA,\ VP \to V\ NPA\ PP,$
$V \to \textit{chowa},\ PP \to \textit{do}\ NPG,\ NPN \to NN,\ NPN \to NN\ PP,$
$NPG \to NG,\ NPG \to NG\ PP,\ NPA \to NA,\ NPA \to NA\ PP,$
$NN \to \textit{szewc},\ NN \to \textit{pasta},\ NN \to \textit{buty},$
$NG \to \textit{szewca},\ NG \to \textit{pasty},\ NG \to \textit{butów},$
$NA \to \textit{szewca},\ NA \to \textit{pastę},\ NA \to \textit{buty} \}$,
— symbol początkowy $S \in V$,
— wartości prawdopodobieństw reguł:
$P(S \to NPN\ VP) = 1$,
$P(VP \to V\ NPA) = 0.75$,
$P(VP \to V\ NPA\ PP) = 0.25$,
$P(V \to \textit{chowa}) = 1$,
$P(PP \to \textit{do}\ NPG) = 1$,
$P(NPN \to NN) = 0.65$,
$P(NPN \to NN\ PP) = 0.35$,
$P(NPG \to NG) = 0.7$,
$P(NPG \to NG\ PP) = 0.3$,
$P(NPA \to NA) = 0.6$,
$P(NPA \to NA\ PP) = 0.4$,
$P(NN \to \textit{szewc}) = 0.4$,
$P(NN \to \textit{pasta}) = 0.3$,
$P(NN \to \textit{buty}) = 0.3$,
$P(NG \to \textit{szewca}) = 0.3$,
$P(NG \to \textit{pasty}) = 0.3$,

$P(\ NG\ \rightarrow\ butów\ ) = 0.4,$
$P(\ NA\ \rightarrow\ szewca\ ) = 0.25,$
$P(\ NA\ \rightarrow\ pastę\ ) = 0.3,$
$P(\ NA\ \rightarrow\ buty\ ) = 0.45.$

Przykład 2.26 (zbinaryzowana probabilistyczna gramatyka bezkontekstowa)**.** Oto przykład zbinaryzowanej probabilistycznej gramatyki bezkontekstowej:

— $G = (V, T, R, S, P)$,
— $V = \{\ S\ ,\ NN\ ,\ NA\ ,\ NP\ ,\ AN\ ,\ V\ ,\ VP\ \}$,
— $T = \{\ gra\ ,\ grę\ ,\ kiera\ ,\ nieznajoma\ ,\ nieznajomą\ ,\ piękna\ ,\ pika\ ,\ pikę\ \}$,
— $R = \{\ S\ \rightarrow\ NP\ \ VP\ ,\ NP\ \rightarrow\ NN\ ,\ NP\ \rightarrow\ AN\ \ NN\ ,$
$VP\ \rightarrow\ V\ ,\ VP\ \rightarrow\ V\ \ NA\ ,$
$NN\ \rightarrow\ gra\ ,\ NN\ \rightarrow\ nieznajoma\ ,\ NN\ \rightarrow\ pika\ ,\ NA\ \rightarrow\ grę\ ,$
$NA\ \rightarrow\ nieznajomą\ ,\ NA\ \rightarrow\ pikę\ ,\ NA\ \rightarrow\ pika\ ,\ NA\ \rightarrow\ kiera\ ,$
$V\ \rightarrow\ gra\ ,\ V\ \rightarrow\ pika\ ,\ AN\ \rightarrow\ piękna\ ,\ AN\ \rightarrow\ nieznajoma\ \}$,
— symbol początkowy $S\ \in V$,
— wartości prawdopodobieństw reguł:
$P(\ S\ \rightarrow\ NP\ \ VP\ ) = 1,$
$P(\ NP\ \rightarrow\ NN\ ) = 0.7,$
$P(\ NP\ \rightarrow\ AN\ \ NN\ ) = 0.3,$
$P(\ VP\ \rightarrow\ V\ ) = 0.4,$
$P(\ VP\ \rightarrow\ V\ \ NA\ ) = 0.6,$
$P(\ NN\ \rightarrow\ gra\ ) = 0.5.$
$P(\ NN\ \rightarrow\ nieznajoma\ ) = 0.2.$
$P(\ NN\ \rightarrow\ pika\ ) = 0.3.$
$P(\ NA\ \rightarrow\ grę\ ) = 0.3.$
$P(\ NA\ \rightarrow\ nieznajomą\ ) = 0.15.$
$P(\ NA\ \rightarrow\ pikę\ ) = 0.15.$
$P(\ NA\ \rightarrow\ pika\ ) = 0.2.$
$P(\ NA\ \rightarrow\ kiera\ ) = 0.2.$
$P(\ V\ \rightarrow\ gra\ ) = 0.8.$
$P(\ V\ \rightarrow\ pika\ ) = 0.2.$
$P(\ AN\ \rightarrow\ piękna\ ) = 0.7.$
$P(\ AN\ \rightarrow\ nieznajoma\ ) = 0.3.$

Przykład 2.27 (probabilistyczna gramatyka bezkontekstowa w postaci normalnej Chomsky'ego)**.** Poniższa probabilistyczna gramatyka bezkontekstowa jest w postaci normalnej Chomsky'ego:

— $G = (V, T, R, S, P)$,
— $V = \{\ S\ ,\ NN\ ,\ NA\ ,\ NP\ ,\ AN\ ,\ V\ ,\ VP\ \}$,
— $T = \{\ gra\ ,\ grę\ ,\ kiera\ ,\ nieznajoma\ ,\ nieznajomą\ ,\ piękna\ ,\ pika\ ,\ pikę\ \}$,
— $R = \{\ S\ \rightarrow\ NP\ \ VP\ ,\ S\ \rightarrow\ NP\ \ V\ ,\ S\ \rightarrow\ NN\ \ VP\ ,\ S\ \rightarrow\ NN\ \ V\ ,$
$NP\ \rightarrow\ AN\ \ NN\ ,\ VP\ \rightarrow\ V\ \ NA\ ,$
$NN\ \rightarrow\ gra\ ,\ NN\ \rightarrow\ nieznajoma\ ,\ NN\ \rightarrow\ pika\ ,\ NA\ \rightarrow\ grę\ ,$
$NA\ \rightarrow\ nieznajomą\ ,\ NA\ \rightarrow\ pikę\ ,\ NA\ \rightarrow\ pika\ ,\ NA\ \rightarrow\ kiera\ ,$
$V\ \rightarrow\ gra\ ,\ V\ \rightarrow\ pika\ ,\ AN\ \rightarrow\ piękna\ ,\ AN\ \rightarrow\ nieznajoma\ \}$,
— symbol początkowy $S\ \in V$,
— wartości prawdopodobieństw reguł:
$P(\ S\ \rightarrow\ NP\ \ VP\ ) = 0.18,$
$P(\ S\ \rightarrow\ NP\ \ V\ ) = 0.12,$

$P(\ S\ \rightarrow\ NN\ \ VP\ ) = 0.42,$
$P(\ S\ \rightarrow\ NN\ \ V\ ) = 0.28,$
$P(\ NP\ \rightarrow\ AN\ \ NN\ ) = 1,$
$P(\ VP\ \rightarrow\ V\ \ NA\ ) = 1,$
$P(\ NN\ \rightarrow\ gra\ ) = 0.5.$
$P(\ NN\ \rightarrow\ nieznajoma\ ) = 0.2.$
$P(\ NN\ \rightarrow\ pika\ ) = 0.3.$
$P(\ NA\ \rightarrow\ grę\ ) = 0.3.$
$P(\ NA\ \rightarrow\ nieznajomą\ ) = 0.15.$
$P(\ NA\ \rightarrow\ pikę\ ) = 0.15.$
$P(\ NA\ \rightarrow\ pika\ ) = 0.2.$
$P(\ NA\ \rightarrow\ kiera\ ) = 0.2.$
$P(\ V\ \rightarrow\ gra\ ) = 0.8.$
$P(\ V\ \rightarrow\ pika\ ) = 0.2.$
$P(\ AN\ \rightarrow\ piękna\ ) = 0.7.$
$P(\ AN\ \rightarrow\ nieznajoma\ ) = 0.3.$

Gramatyka ta powstała przez zmodyfikowanie gramatyki z przykładu 2.26 w taki sposób, żeby uzyskać równoważną jej gramatykę w postaci normalnej Chomsky'ego.

Zanim zdefiniujemy prawdopodobieństwo łańcucha, musimy zastanowić się, jak określić prawdopodobieństwo poszczególnego jego drzewa rozkładu (lub, równoważnie, wyprowadzenia lewostronnego). Niech

$$S = \zeta_0 \Rightarrow \zeta_1 \Rightarrow \ldots \Rightarrow \zeta_m = w$$

będzie wyprowadzeniem lewostronnym łańcucha w. Przyjmijmy, że (dla każdego i) łańcuch ζ_i wyprowadzono bezpośrednio z łańcucha ζ_{i-1} wykorzystując regułę $r_i \in R$. Prawdopodobieństwo, że w procedurze wyprowadzania otrzymamy łańcuch w z symbolu początkowego S można wyliczyć jako:

$$\begin{aligned}\mathbf{P}(w|S) = \mathbf{P}(r_m|\zeta_{m-1}) \cdot \mathbf{P}(r_{m-1}|\zeta_{m-2}) \cdot \ldots \cdot \mathbf{P}(r_2|\zeta_1) \cdot \mathbf{P}(r_1|\zeta_0) = \\ = P(r_m) \cdot P(r_{m-1}) \cdot \ldots \cdot P(r_2) \cdot P(r_1).\end{aligned}$$

Stąd następująca definicja:

Definicja 2.53 (prawdopodobieństwo wyprowadzenia lewostronnego)**.** *Prawdopodobieństwem wyprowadzenia lewostronnego* nazywamy iloczyn prawdopodobieństw wszystkich reguł użytych w tym wyprowadzeniu lewostronnym.

Ponieważ każdemu drzewu wyprowadzenia możemy jednoznacznie przyporządkować wyprowadzenie lewostronne, prawdopodobieństwo drzewa wyprowadzenia definiujemy następująco:

Definicja 2.54 (prawdopodobieństwo drzewa rozkładu)**.** *Prawdopodobieństwo drzewa rozkładu* definiujemy jako prawdopodobieństwo wyprowadzenia lewostronnego odpowiadającego temu drzewu.

Analogicznie do definicji prawdopodobieństwa wyprowadzenia lewostronnego możemy zdefiniować prawdopodobieństwo dowolnego wyprowadzenia:

Definicja 2.55 (prawdopodobieństwo wyprowadzenia)**.** *Prawdopodobieństwem wyprowadzenia* nazywamy iloczyn prawdopodobieństw wszystkich reguł użytych w tym wyprowadzeniu.

Ponieważ różne wyprowadzenia odpowiadające jednemu drzewu wyprowadzenia mogą różnić się co najwyżej kolejnością stosowania reguł, zatem każde z nich ma takie samo prawdopodobieństwo.

Prawdopodobieństwo dowolnego wyprowadzenia nie jest tak często używane, jak prawdopodobieństwo drzewa (równoważnie: wyprowadzenia lewostronnego), ponieważ nie posiada równie dobrych własności. Na przykład suma prawdopodobieństw wszystkich wyprowadzeń danego łańcucha może być większa od 1, podczas gdy suma prawdopodobieństw wyprowadzeń lewostronnych łańcucha nigdy nie przekracza 1.

Ten ostatni fakt wykorzystuje się do zdefiniowania prawdopodobieństwa łańcucha:

Definicja 2.56 (prawdopodobieństwo łańcucha)**.** *Prawdopodobieństwem łańcucha* nazywamy sumę prawdopodobieństw wszystkich wyprowadzeń lewostronnych tego łańcucha (w danej gramatyce).

Powyższą definicję interpretuje się w ten sposób, że prawdopodobieństwo, iż łańcuch w wyprowadzimy z symbolu początkowego, jest sumą prawdopodobieństw wszystkich sposobów, na jakie możemy to zrobić.

Definicja 2.57 (drzewo Viterbiego)**.** *Drzewami Viterbiego* łańcucha w nazywamy te spośród wszystkich drzew wyprowadzeń łańcucha w, których prawdopodobieństwa są największe.

Definicja 2.58 (prawdopodobieństwo Viterbiego)**.** *Prawdopodobieństwem Viterbiego* łańcucha w nazywamy prawdopodobieństwo jego drzewa Viterbiego. Prawdopodobieństwo Viterbiego łańcucha w oznaczamy przez $\nu(w)$.

Przykład 2.28 (wyprowadzenie łańcucha w probabilistycznej gramatyce bezkontekstowej)**.** Rysunek 2.6 pokazuje dwa możliwe drzewa wyprowadzenia łańcucha $w =$ *szewc chowa pastę do butów* w gramatyce probabilistycznej z przykładu 2.25.

Prawdopodobieństwo drzewa wyprowadzenia Γ wynosi

$$\mathbf{P}(\Gamma) = 1 \cdot 0.65 \cdot 0.4 \cdot 0.25 \cdot 1 \cdot 0.6 \cdot 0.3 \cdot 1 \cdot 0.7 \cdot 0.4 = 0.003276.$$

Prawdopodobieństwo drzewa wyprowadzenia Δ wynosi

$$\mathbf{P}(\Delta) = 1 \cdot 0.65 \cdot 0.4 \cdot 0.75 \cdot 1 \cdot 0.4 \cdot 0.3 \cdot 1 \cdot 0.7 \cdot 0.4 = 0.006552.$$

Drzewa Γ i Δ są jedynymi drzewami wyprowadzenia łańcucha w w gramatyce G. Prawdopodobieństwo łańcucha w wynosi zatem

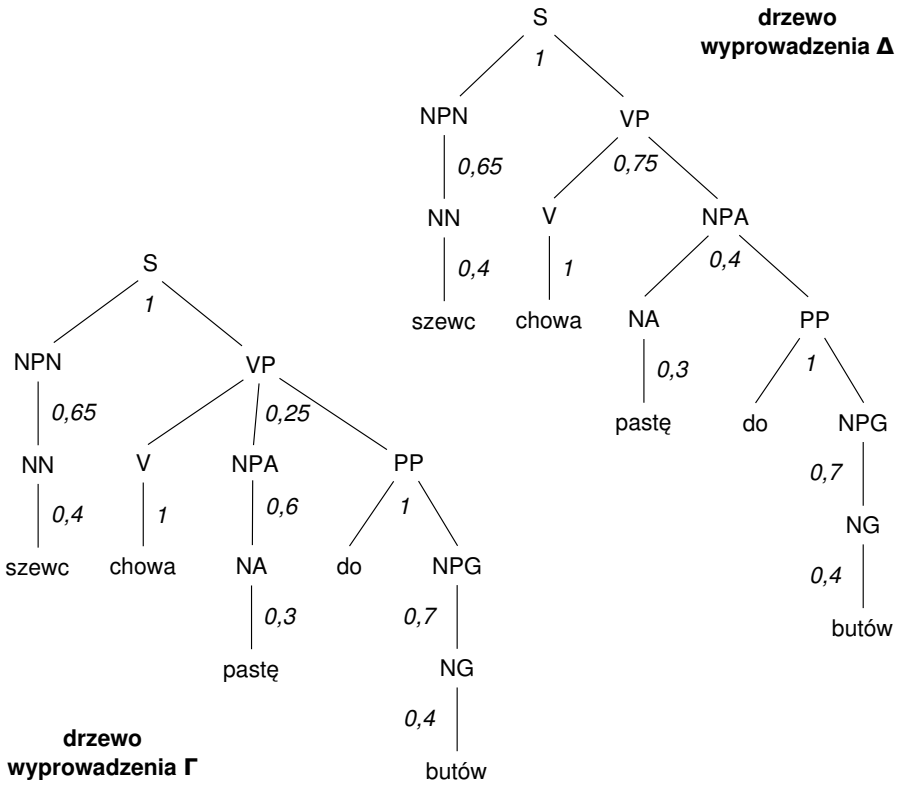
$$P(w) = \mathbf{P}(\Gamma) + \mathbf{P}(\Delta) = 0.003276 + 0.006552 = 0.009828.$$

Drzewem Viterbiego łańcucha w jest drzewo Δ, ponieważ ma większe prawdopodobieństwo niż drzewo Γ. Prawdopodobieństwo Viterbiego łańcucha w wynosi zatem

$$\nu(w) = \mathbf{P}(\Delta) = 0.006552.$$

Definicja 2.59 (prawdopodobieństwo zewnętrzne)**.** Niech $G = (V, T, R, S, P)$ będzie probabilistyczną gramatyką bezkontekstową oraz Γ niech będzie drzewem wyprowadzenia łańcucha $w = w_1 \dots w_n \in L(G)$. *Prawdopodobieństwo zewnętrzne* łańcucha $\zeta \in (V \cup T)^+$ w zakresie (i, j), gdzie $0 \leqslant i \leqslant j \leqslant n$ definiujemy jako

$$\alpha(\zeta[i, j]) := \mathbf{P}(w_1 \dots w_i \zeta w_{j+1} \dots w_n).$$



Rysunek 2.6. Dwa różne drzewa rozkładu łańcucha *szewc chowa pastę do butów* .

Definicja 2.60 (prawdopodobieństwo wewnętrzne)**.** Niech $G = (V, T, R, S, P)$ będzie probabilistyczną gramatyką bezkontekstową oraz Γ niech będzie drzewem wyprowadzenia łańcucha $w = w_1 \dots w_n \in L(G)$. *Prawdopodobieństwo wewnętrzne* łańcucha $\zeta \in (V \cup T)^+$ w zakresie (i, j), gdzie $0 \leqslant i \leqslant j \leqslant n$ definiujemy jako

$$\beta(\zeta[i, j]) := \mathbf{P}(w_{i+1} \dots w_j | \zeta).$$

Rozdział 3

Przedstawienie istniejących algorytmów parsingu probabilistycznych gramatyk bezkontekstowych

3.1. Ogólna charakterystyka podstawowych metod parsingu gramatyk bezkontekstowych

3.1.1. Parsing zstępujący i wstępujący[1]

Parsing to jedno z podstawowych pojęć analizy języka naturalnego. Oznacza ono procedurę rozpoznawania, czy dany łańcuch (zdanie) należy do danego języka, połączoną z przypisaniem temu łańcuchowi pewnej (związanej z gramatyką) struktury. Bliskim odpowiednikiem pojęcia *parsing* jest określenie *analiza składniowa*, jednak mówiąc o analizie składniowej, myślimy na ogół o procesie niezmechanizowanym, rozumianym w kontekście językoznawczym raczej niż informatycznym. Parsing natomiast najczęściej rozumiemy jako procedurę automatyczną, wykonywaną przez komputer.

W przypadku gramatyki bezkontekstowej oczekujemy od algorytmu parsującego, czyli *parsera*, żeby sprawdził, czy dany łańcuch należy do języka generowanego przez tę gramatykę, oraz znalazł drzewo (drzewa) wyprowadzenia tego łańcucha. Celem parsera może być znalezienie wszystkich możliwych drzew wyprowadzenia bądź tylko jednego — dowolnego lub spełniającego określone warunki. Z praktycznego punktu widzenia najlepiej jest, jeśli znalezione drzewo (drzewa) odzwierciedla spodziewane znaczenie parsowanego zdania. W przypadku probabilistycznych gramatyk bezkontekstowych zakłada się na ogół, że ten warunek spełnia drzewo Viterbiego, dlatego znalezienie drzewa Viterbiego dla danego łańcucha jest jednym z głównych zadań parserów gramatyk probabilistycznych.

Wyróżnia się dwie główne strategie parsowania gramatyk bezkontekstowych: parsowanie zstępujące (ang. *top-down*) i parsowanie wstępujące (ang. *bottom-up*).

Podejście zstępujące polega na tym, że budowanie drzewa rozkładu wejściowego łańcucha rozpoczyna się od symbolu początkowego gramatyki. W kolejnych krokach algorytmu rozwija się kolejne symbole pomocnicze, konstruując coraz bardziej rozbudowane drzewa, dopóki nie dotrze się do symboli końcowych. Jeżeli po drodze okaże się, że pewne wyprowadzenia nie pasują do symboli łańcucha wejściowego, odrzuca się je.

Podejście wstępujące działa w odmienny sposób. Budowę drzewa wyprowadzenia łańcucha wejściowego rozpoczyna się od tegoż łańcucha. Najpierw szuka się produkcji, za pomocą których można wyprowadzić symbole końcowe łańcucha wejściowego. W kolejnych krokach algorytmu łączy się powstałe w ten

[1] Na podstawie [7].

sposób małe poddrzewa za pomocą reguł produkcji w coraz większe, dopóki nie osiągnie się symbolu początkowego gramatyki.

Zarówno jedno, jak i drugie podejście ma swoje zalety, ale też nie pozbawione jest wad. Zastosowanie strategii zstępującej pozwala uniknąć konstruowania drzew, których nie można wywieść z symbolu początkowego gramatyki. Z kolei podejście wstępujące pozwala uniknąć przeszukiwania wyprowadzeń, które nie prowadzą do łańcucha wejściowego. Dlatego współczesne algorytmy parsingu stosują na ogół strategie mieszane, czego dobrym przykładem może być parser Earleya (patrz sekcja 3.2).

3.1.2. Programowanie dynamiczne i parsing tablicowy[2]

Podejściem często wykorzystywanym w algorytmach parsowania gramatyk bezkontekstowych jest *programowanie dynamiczne*. Programowanie dynamiczne jest metodą programowania stosowaną, gdy zadanie można podzielić na podproblemy, które niekoniecznie są rozłączne ze sobą. Wówczas rozwiązuje się najpierw poszczególne podproblemy, a ich rozwiązania składuje się w tablicy. Korzysta się następnie z tych rozwiązań do rozwiązania większych podproblemów, a ostatecznie — całego zadania.

W kontekście parsowania podproblemami mogą być zadania przeprowadzenia analizy składniowej dla poszczególnych podłańcuchów łańcucha wejściowego. Drzewa wyprowadzeń mniejszych podłańcuchów można ze sobą łączyć, tworząc drzewa rozkładu dla większych podłańcuchów.

Programowanie dynamiczne często jest wykorzystywane w algorytmach parsingu gramatyk bezkontekstowych, np. w algorytmie Earleya i w algorytmie CYK (patrz sekcja 3.3).

Pojęcie programowania dynamicznego jest silnie związane z pojęciem *parsingu tablicowego* (ang. *chart parsing*). Mianem parsera tablicowego określa się często każdy algorytm parsingu, który wykorzystuje tablicę do przechowywania wyników cząstkowych uzyskanych zgodnie z paradygmatem programowania dynamicznego (w tym m.in. algorytm Earleya i algorytm CYK).

W węższym znaczeniu, parserem tablicowym nazywa się taką implementację algorytmu parsingu, która wykorzystuje trzy następujące struktury danych:

— *tablicę* elementów składających się z symbolu pomocniczego gramatyki oraz jego zakresu w drzewie wyprowadzenia,
— *agendę* — strukturę przechowującą elementy (zdefiniowane powyżej), które nie zostały jeszcze dodane do tablicy,
— *zbiór krawędzi*, gdzie *krawędź* należy rozumieć jako regułę z dołączonymi znacznikami mówiącymi o tym, w którym miejscu drzewa została zastosowana i jaka jej część została już przeanalizowana (jest to pojęcie podobne do pojęcia reguły z kropką stosowanej w algorytmie Earleya).

W trakcie działania algorytmu krawędzie przetwarzane są od lewej do prawej, w miarę dopasowywania coraz to nowych elementów do symboli znajdujących się po prawej stronie reguły krawędzi. Kiedy wszystkie symbole po prawej stronie reguły krawędzi zostaną dopasowane, krawędź oznaczana jest jako *kompletna*.

Ogólny schemat działania parsera tablicowego przedstawiony jest poniżej.

[2] Na podstawie [1] i [7].

Algorytm 3.1 (parsing tablicowy gramatyk bezkontekstowych[3])**.** Niech dana będzie gramatyka bezkontekstowa $G = (V, T, R, S)$ oraz łańcuch $w = w_1 \dots w_n \in T^+$.

1. Przygotuj pustą tablicę.
2. Przygotuj pusty zbiór krawędzi.
3. Przygotuj agendę i włóż do niej elementy $w_1[0,1]$, $w_2[1,2]$, …, $w_n[n-1,n]$.
4. Powtarzaj dopóki agenda nie będzie pusta:
   a) wyjmij z agendy kolejny element;
   b) jeżeli tablica zawiera już identyczny element, wróć do kroku 4a;
   c) dodaj ten element do tablicy;
   d) dla każdej reguły gramatyki G wykonaj:
      — jeżeli pierwszy symbol po prawej stronie reguły pasuje do przetwarzanego elementu, to utwórz w zbiorze krawędzi nową krawędź na bazie tej reguły;
   e) dla każdej niekompletnej krawędzi ze zbioru krawędzi wykonaj:
      i. jeżeli pierwszy niedopasowany symbol po prawej stronie reguły krawędzi pasuje do przetwarzanego elementu, to dopasuj ten symbol do tego elementu i zaktualizuj odpowiednio znaczniki,
      ii. jeżeli krawędź stała się przez to kompletną, to dodaj symbol po lewej stronie reguły wraz z jego zakresem jako nowy element agendy i usuń kompletną krawędź ze zbioru krawędzi.
5. Jeżeli $w \in L(G)$, to po zakończeniu działania algorytmu tablica zawiera strukturę drzewa wyprowadzenia wejściowego łańcucha.

Kolejność, w jakiej elementy są wyjmowane z agendy, może być różna, w zależności od wariantu algorytmu. Agenda może być zaimplementowana jako prosty stos lub kolejka, ale może być też zaawansowaną kolejką priorytetową o priorytecie wyznaczonym przez różne funkcje oceny.

Odmiany parserów tablicowych zostaną przedstawione w sekcjach 3.4.2 i 3.4.3 oraz w rozdziale 4.

Twierdzenie 3.1 (złożoność czasowa parsingu tablicowego[4])**.** *Złożoność czasowa algorytmu parsingu tablicowego wynosi $O(n^3)$ dla łańcucha wejściowego o długości n.*

Dowód. Każda krawędź jest przetwarzana dokładnie raz. Każda krawędź charakteryzowana jest przez cztery parametry:
— symbol — liczba symboli nie zależy od długości n wejściowego łańcucha,
— początek zakresu — ograniczony przez $O(n)$,
— koniec zakresu — ograniczony przez $O(n)$,
— postęp przetworzenia — również ograniczony przez $O(n)$.
Z tego powodu łączna liczba przetworzonych krawędzi ograniczona jest przez $O(n^3)$, a zatem złożoność czasowa algorytmu wynosi $O(n^3)$. □

[3] Na podstawie [4].
[4] Na podstawie [8].

3.2. Algorytm Earleya

3.2.1. Algorytm Earleya dla gramatyk bezkontekstowych[5]

Do parsowania probabilistycznych gramatyk bezkontekstowych można wykorzystać zmodyfikowany *algorytm Earleya*.

W algorytmie Earleya dla gramatyk bezkontekstowych symbole łańcucha wejściowego $w = w_1 \dots w_n$ przetwarzane są kolejno od lewej do prawej. Wykorzystywana jest w tym celu $(n+1)$-elementowa tablica, która zapełniana jest sukcesywnie od lewej do prawej w czasie działania algorytmu. Dla każdej pozycji symbolu w łańcuchu w, tablica zawiera listę stanów (tzw. *reguł z kropką*) reprezentujących wygenerowane do tej pory częściowe drzewa rozkładu. Gdy algorytm dociera do końca łańcucha w, tablica zawiera zakodowane wszystkie możliwe drzewa wyprowadzeń tego łańcucha.

Reguła z kropką jest specjalną strukturą danych wykorzystywaną w parserze Earleya. Zawiera ona informacje o poddrzewie odpowiadającym pojedynczej regule gramatyki, o jego pozycji względem łańcucha wejściowego oraz o tym, jaką część poddrzewa udało się już skompletować w czasie działania algorytmu. Dla gramatyki bezkontekstowej $G = (V, T, R, S)$ i łańcucha wejściowego $w = w_1 \dots w_n$ ogólna postać reguły z kropką przedstawia się następująco:

$$A \to \zeta \bullet \xi[i, j],$$

gdzie $A \in V$, $\zeta, \xi \in (V \cup T)^*$, $(A \to \zeta\xi) \in R$, $i, j \in \{0, 1, \dots n\}$. $A \to \zeta\xi$ jest aktualnie przetwarzaną regułą gramatyki G. Po lewej stronie kropki (łańcuch ζ, na lewo od $\bullet$) znajdują się symbole już dopasowane do symboli wejściowego łańcucha. Po prawej (łańcuch ξ) — te, które jeszcze nie zostały przetworzone. Liczba i oznacza miejsce, w którym został dopasowany początek reguły, zaś j — aktualną pozycję w przetwarzanym wejściu, czyli liczbę przetworzonych symboli łańcucha wejściowego. Inaczej mówiąc, symbole $w_1 \dots w_i$ zostały już przetworzone, a $w_{i+1} \dots w_j$ dopasowano do łańcucha ζ.

Aby algorytm Earleya pozwolił nie tylko sprawdzić, czy dany łańcuch należy do języka generowanego przez gramatykę, ale też znaleźć jego drzewa rozkładu, należy dołączyć do każdej reguły z kropką pole, w które będzie można w razie potrzeby wstawić wskaźnik do innej reguły z kropką.

Główną częścią algorytmu Earleya są trzy procedury, które są stosowane w zależności od kształtu aktualnie przetworzonej reguły z kropką — przewidywanie, wczytywanie i uzupełnianie. Każda z tych procedur powoduje dodanie do tablicy nowych reguł z kropką.

Niech t będzie tablicą używaną w algorytmie, niech listy $t_0, t_1, \dots, t_n$ będą elementami tej tablicy. Niech $w = w_1 w_2 \dots w_n$ będzie łańcuchem na wejściu algorytmu.

Procedura 3.2 (przewidywanie)**.** Argumentem procedury jest reguła z kropką postaci

$$A \to \zeta \bullet B\xi[i, j],$$

gdzie $A, B \in V$, $\zeta, \xi \in (V \cup T)^*$, $i, j \in \{0, 1, \dots n\}$.

1. Dla każdej reguły gramatyki postaci $B \to \omega$, $\omega \in (V \cup T)^+$:

[5] Na podstawie [7].

— zakolejkuj na liście t_j regułę z kropką

$$B \to \bullet\omega[j,j].$$

Procedura 3.3 (wczytywanie)**.** Argumentem procedury jest reguła z kropką postaci

$$A \to \zeta \bullet a\xi[i,j],$$

gdzie $A \in V$, $\zeta, \xi \in (V \cup T)^*$, $a \in T$, $i, j \in \{0, 1, \ldots n\}$.

1. Jeżeli $a = w_j$, to:
   — zakolejkuj na liście t_{j+1} regułę z kropką

   $$A \to \zeta a \bullet \xi[i, j+1]$$

   (czyli $A \to \zeta w_j \bullet \xi[i, j+1]$).

Procedura 3.4 (uzupełnianie)**.** Argumentem procedury jest reguła z kropką postaci

$$B \to \omega \bullet [j,k],$$

gdzie $B \in V$, $\omega \in (V \cup T)^+$, $j, k \in \{0, 1, \ldots n\}$.

1. Dla każdej reguły z kropką postaci $A \to \zeta \bullet B\xi[i,j]$ znajdującej się na liście t_j wykonaj:
   a) utwórz regułę z kropką

   $$A \to \zeta B \bullet \xi[i,k],$$

   b) dodaj do tej reguły wskaźnik do reguły z kropką, która była argumentem procedury,
   c) zakolejkuj tę regułę na liście t_k.

Przez zakolejkowanie reguły z kropką na liście rozumiemy dodanie tej reguły z kropką na końcu listy pod warunkiem, że jeszcze nie znalazła się na tej liście.

Cały algorytm dla gramatyk nieprobabilistycznych przedstawia się następująco:

Algorytm 3.5 (algorytm Earleya parsowania gramatyk bezkontekstowych)**.** Niech dana będzie gramatyka bezkontekstowa $G = (V, T, R, S)$ oraz łańcuch $w = w_1 w_2 \ldots w_n$.

1. Zainicjalizuj tablicę $t = (t_0, t_1, \ldots t_n)$ pustymi listami.
2. Zakolejkuj na liście t_0 specjalną regułę z kropką

   $$\gamma \to \bullet S[0,0].$$

3. Dla l od 0 do n wykonuj:
   — dla każdej reguły z kropką z listy t_l wykonuj:
   a) jeśli reguła z kropką jest postaci

   $$A \to \zeta \bullet B\xi[i,l],$$

   gdzie $A, B \in V$, $\zeta, \xi \in (V \cup T)^*$, $i \in \{0, 1, \ldots l\}$, to wykonaj na niej procedurę 3.2 (przewidywanie),
   b) jeśli reguła z kropką jest postaci

   $$A \to \zeta \bullet w_l\xi[i,l],$$

   gdzie $A \in V$, $\zeta, \xi \in (V \cup T)^*$, $i \in \{0, 1, \ldots l\}$, to wykonaj na niej procedurę 3.3 (wczytywanie),

c) jeśli reguła z kropką jest postaci

$$B \rightarrow \omega \bullet [i, l],$$

gdzie $B \in V$, $\omega \in (V \cup T)^+$, $i \in \{0, 1, \ldots l\}$, to wykonaj na niej procedurę 3.4 (uzupełnianie),

4. Obecność reguły z kropką $\gamma \rightarrow S \bullet [0, n]$ na liście t_n oznacza, że łańcuch w należy do języka $L(G)$.
5. Drzewa rozbioru łańcucha w można odtworzyć podążając za wskaźnikami dołączonymi do reguł wskazywanych przez regułę $\gamma \rightarrow S \bullet [0, n]$.

Twierdzenie 3.2. *W ogólnym przypadku złożoność czasowa algorytmu Earleya jest sześcienna, tj. dla łańcucha wejściowego o długości n algorytm działa w czasie* $O(n^3)$.

Dowód. Liczba reguł z kropką na każdej liście t_l jest proporcjonalna do $l = O(n)$. Każda procedura przewidywania wykonuje liczbę kroków ograniczoną przez liczbę reguł gramatyki, a więc niezależną od n. Procedura wczytywania wykonuje pojedynczy krok. Jeżeli argumentem procedury uzupełniania jest reguła z kropką z listy t_k, to procedura ta wykonuje co najwyżej tyle kroków, ile jest reguł na liście t_j, gdzie $j < k$. Wynika stąd, że przetworzenie wszystkich reguł na liście t_l wymaga liczby kroków proporcjonalnej do l^2, czyli $O(n^2)$. Po zsumowaniu kroków potrzebnych do przetworzenia wszystkich list $t_0, t_1, \ldots, t_n$ otrzymujemy złożoność czasową w najgorszym przypadku $O(n^3)$. [3] □

Twierdzenie 3.3. *Złożoność pamięciowa algorytmu Earleya wynosi co najwyżej* $O(n^2)$.

Dowód. Na każdej z $n + 1 = O(n)$ list znajduje się co najwyżej $O(n)$ reguł z kropką, co daje złożoność pamięciową rzędu $O(n^2)$. [3] □

Jeżeli nałoży się dodatkowe ograniczenia na postać gramatyki, złożoność czasową algorytmu można zmniejszyć do $O(n^2)$ (dla gramatyk jednoznacznych) lub nawet do $O(n)$ (dla prawie wszystkich gramatyk klasy LR(k)) [3].

3.2.2. Algorytm Earleya dla probabilistycznych gramatyk bezkontekstowych[6]

Algorytm Earleya znajdowania drzewa Viterbiego dla gramatyk probabilistycznych jest rozszerzeniem opisanego powyżej algorytmu Earleya. Na listach tablicy t oprócz reguł z kropką i przypisanych im wskaźników przechowuje się również *prawdopodobieństwo Viterbiego* każdej takiej reguły, czyli cząstkowe prawdopodobieństwo maksymalne obliczone na danym etapie działania algorytmu.

Kiedy wykonywana jest jedna z trzech procedur algorytmu Earleya, obliczana jest nowa wartość prawdopodobieństwa Viterbiego na podstawie prawdopodobieństw Viterbiego reguł będących argumentami procedur. Obliczona wartość jest dodawana do nowo powstałej reguły, a jeżeli reguła będąca wynikiem procedury znajduje się już w tablicy, to prawdopodobieństwo Viterbiego tej reguły zostaje odpowiednio zaktualizowane: jeżeli nowa wartość jest większa od dotychczasowej, to dotychczasowa wartość zostaje zastąpiona przez nową.

[6] Na podstawie [15].

Gdy algorytm dojdzie do końca łańcucha wejściowego, maksymalne prawdopodobieństwo łańcucha wejściowego jest prawdopodobieństwem Viterbiego ν przypisanym regule z kropką $\gamma \to S \bullet [0, n]$. W celu odszukania drzewa Viterbiego danego łańcucha trzeba na zakończenie działania algorytmu wykonać jeszcze jedną dodatkową operację, mianowicie prześledzić, zastosowanie których reguł dało taką wartość prawdopodobieństwa Viterbiego łańcucha. Za wykonanie tego zadania odpowiada procedura 3.9, opisana poniżej.

Niech $G = (V, T, R, S, P)$ będzie probabilistyczną gramatyką bezkontekstową, zaś $w = w_1 \dots w_n$ — łańcuchem danym na wejściu.

Procedura 3.6 (przewidywanie — wersja probabilistyczna)**.** Argumentem procedury jest reguła z kropką postaci

$$A \to \zeta \bullet B\xi[i, j],$$

gdzie $A, B \in V$, $\zeta, \xi \in (V \cup T)^*$, $i, j \in \{0, 1, \dots n\}$.

1. Dla każdej reguły gramatyki postaci $B \to \omega$, $\omega \in (V \cup T)^+$:
   a) utwórz regułę z kropką
   $$B \to \bullet\omega[j, j],$$
   b) przypisz jej prawdopodobieństwo Viterbiego równe $P(B \to \omega)$,
   c) zakolejkuj na liście t_j tę regułę z kropką (z kroku 1a) wraz z przypisanym jej prawdopodobieństwem Viterbiego (z kroku 1b).

Procedura 3.7 (wczytywanie — wersja probabilistyczna)**.** Argumentem procedury jest reguła z kropką postaci

$$A \to \zeta \bullet a\xi[i, j]$$

($A \in V$, $\zeta, \xi \in (V \cup T)^*$, $a \in T$, $i, j \in \{0, 1, \dots n\}$) o prawdopodobieństwie Viterbiego równym ν_0.

1. Jeżeli $a = w_j$, to:
   a) utwórz regułę z kropką
   $$A \to \zeta a \bullet \xi[i, j + 1],$$
   b) przypisz jej prawdopodobieństwo Viterbiego równe ν_0,
   c) zakolejkuj na liście t_{j+1} tę regułę z kropką (z kroku 1a) wraz z przypisanym jej prawdopodobieństwem Viterbiego (z kroku 1b).

Procedura 3.8 (uzupełnianie — wersja probabilistyczna)**.** Argumentem procedury jest reguła z kropką

$$B \to \omega \bullet [j, k]$$

($B \in V$, $\omega \in (V \cup T)^+$, $j, k \in \{0, 1, \dots n\}$) o prawdopodobieństwie Viterbiego równym ν_0.

1. Dla każdej reguły z kropką r postaci $A \to \zeta \bullet B\xi[i, j]$ (gdzie $A \in V$, $\zeta, \xi \in (V \cup T)^*$, $i \in \{0, 1, \dots, j\}$) znajdującej się na liście t_j wykonaj:
   a) odczytaj prawdopodobieństwo Viterbiego $\nu(r)$ reguły z kropką r,
   b) jeżeli na liście t_k znajduje się już reguła z kropką r' postaci $A \to \zeta B \bullet \xi[i, k]$, to:

i. odczytaj prawdopodobieństwo Viterbiego $\nu(r')$ reguły z kropką r',
ii. jeśli $\nu(r) \cdot \nu_0 > \nu(r')$, to przypisz regule z kropką r' nowe prawdopodobieństwo Viterbiego równe $\nu(r) \cdot \nu_0$;

c) w przeciwnym wypadku:
i. utwórz nową regułę z kropką $A \to \zeta B \bullet \xi[i,k]$,
ii. przypisz jej prawdopodobieństwo Viterbiego równe $\nu(r) \cdot \nu_0$,
iii. zakolejkuj na liście t_k tę regułę z kropką wraz z przypisanym jej prawdopodobieństwem Viterbiego.

Funkcja 3.9 (znajdowanie drzewa Viterbiego)**.** Argumentem funkcji jest reguła z kropką postaci

$$A \to \zeta \bullet \xi[k,i],$$

gdzie $A \in V$, $\zeta, \xi \in (V \cup T)^*$, $k, i \in \{0, 1, \ldots, n\}$.

Wartością zwracaną przez funkcję jest drzewo rozkładu.

1. Jeżeli ζ jest łańcuchem pustym, to zwróć jako wynik drzewo złożone z samego tylko korzenia A (bez żadnych synów).
2. W przeciwnym wypadku, jeżeli ostatnim symbolem łańcucha ζ jest symbol końcowy a (czyli $\zeta = \zeta' a$, $\zeta' \in (V \cup T)^*$):
   a) znajdź drzewo Viterbiego Γ dla reguły z kropką

   $$A \to \zeta' \bullet a\xi[k, i-1]$$

   (wywołując tę funkcję rekurencyjnie),
   b) do korzenia drzewa Γ dołącz nowego syna o etykiecie a jako położonego najbardziej na prawo,
   c) zwróć powstałe w ten sposób drzewo jako wynik działania funkcji.
3. W przeciwnym wypadku, jeżeli ostatnim symbolem łańcucha ζ jest symbol pomocniczy B (czyli jeśli $\zeta = \zeta' B$, $\zeta' \in (V \cup T)^*$):
   a) podążając za wskaźnikiem, znajdź regułę z kropką $B \to \omega \bullet [j,i]$, $\omega \in (V \cup T)^*$, której uzupełnienie spowodowało utworzenie reguły z kropką $A \to \zeta' B \bullet \xi[k,i]$,
   b) znajdź drzewo Viterbiego Γ dla reguły z kropką

   $$A \to \zeta' B \bullet \xi[k,i]$$

   (wywołując tę funkcję rekurencyjnie),
   c) znajdź drzewo Viterbiego Δ dla reguły z kropką

   $$B \to \omega \bullet [j,i]$$

   (wywołując tę funkcję rekurencyjnie),
   d) do drzewa Γ dołącz drzewo Δ jako poddrzewo w ten sposób, żeby korzeń drzewa Δ stał się położonym najbardziej na prawo synem korzenia drzewa Γ,
   e) zwróć powstałe w ten sposób drzewo jako wynik działania funkcji.

Oto cały algorytm Earleya znajdowania drzewa Viterbiego:

Algorytm 3.10 (algorytm Earleya parsowania probabilistycznych gramatyk bezkontekstowych)**.** Niech dana będzie probabilistyczna gramatyka bezkontekstowa $G = (V, T, R, S, P)$ oraz łańcuch $w = w_1 w_2 \ldots w_n$.

1. Zainicjalizuj tablicę $t = (t_0, t_1, \ldots, t_n)$ pustymi listami.

2. Zakolejkuj na liście t_0 specjalną regułę z kropką

$$\gamma \rightarrow \bullet S[0,0]$$

i przypisz jej prawdopodobieństwo Viterbiego równe 1.

3. Dla l od 0 do n wykonuj:
   — dla każdej reguły z kropką z listy t_l wykonuj:
   a) jeśli reguła z kropką jest postaci

$$A \rightarrow \zeta \bullet B\xi[i,l],$$

gdzie $A, B \in V$, $\zeta, \xi \in (V \cup T)^*$, $i \in \{0, 1, \ldots l\}$, to wykonaj na niej procedurę 3.6 (przewidywanie),

   b) jeśli reguła z kropką jest postaci

$$A \rightarrow \zeta \bullet w_l\xi[i,l],$$

gdzie $A \in V$, $\zeta, \xi \in (V \cup T)^*$, $i \in \{0, 1, \ldots l\}$, to wykonaj na niej procedurę 3.7 (wczytywanie),

   c) jeśli reguła z kropką jest postaci

$$B \rightarrow \omega \bullet [i,l],$$

gdzie $B \in V$, $\omega \in (V \cup T)^+$, $i \in \{0, 1, \ldots l\}$, to wykonaj na niej procedurę 3.8 (uzupełnianie),

4. Prawdopodobieństwo Viterbiego reguły z kropką $\gamma \rightarrow S \bullet [0,n]$ znajdującej się na liście t_n jest szukanym prawdopodobieństwem drzewa Viterbiego łańcucha w.
5. Drzewo Viterbiego łańcucha w można odtworzyć stosując do reguły $\gamma \rightarrow S \bullet [0,n]$ funkcję 3.9 (znajdowanie drzewa Viterbiego).

Twierdzenie 3.4. *Złożoność czasowa probabilistycznej wersji algorytmu Earleya nie różni się od złożoności wersji nieprobabilistycznej i wynosi w ogólnym przypadku $O(n^3)$ dla łańcucha wejściowego o długości n.*

Dowód. Struktura algorytmu Earleya dla gramatyk probabilistycznych jest niemal identyczna jak dla gramatyk nieprobabilistycznych. Dominującym elementem algorytmu jest nadal procedura uzupełniania, która składa się z $O(l^2)$ kroków dla każdej listy t_l. Dlatego łączna złożoność czasowa pozostaje $O(n^3)$ w najgorszym przypadku. [15] □

Twierdzenie 3.5. *Zależność złożoności czasowej algorytmu Earleya od rozmiaru gramatyki jest również sześcienna i wynosi $O(m^3)$ dla gramatyki o m symbolach pomocniczych.*

Dowód. Najistotniejszym składnikiem obliczeń jest procedura uzupełniania, podczas której muszą być obliczone prawdopodobieństwa dla $O(m^3)$ reguł z kropką w najgorszym przypadku. Stąd pesymistyczna złożoność czasowa całego algorytmu wynosi $O(m^3)$. [15] □

3.3. Algorytm CYK[7]

3.3.1. Algorytm CYK dla gramatyk bezkontekstowych

Algorytm Cocke'a-Youngera-Kasamiego, nazywany w skrócie *algorytmem CYK*, jest algorytmem programowania dynamicznego służącym do parsowania gramatyk bezkontekstowych w postaci normalnej Chomsky'ego.

Nieznacznie zmodyfikowany algorytm CYK może być użyty również do parsowania probabilistycznych gramatyk bezkontekstowych.

Zauważmy, że jeżeli łańcuch ω można rozłożyć na dwa podłańcuchy $\omega = \zeta\xi$, to ich drzewa wyprowadzeń (niekoniecznie z symbolu początkowego gramatyki) można wykorzystać do znalezienia drzewa wyprowadzenia łańcucha ω.

W trakcie działania algorytmu będziemy wykorzystywać tablicę trójwymiarową o wymiarach $n \times n \times m$, gdzie n jest długością parsowanego łańcucha $w = w_1 \dots w_n$, a m jest liczbą symboli nieterminalnych gramatyki. W (i, j, A)-tej komórce tej tablicy będziemy przechowywać informację o tym, czy z symbolu A można wyprowadzić podłańcuch $w_i w_{i+1} \dots w_j$ łańcucha w, oraz wskaźniki do poddrzew, na jakie może zostać rozłożone drzewo wyprowadzenia tego podłańcucha.

Algorytm 3.11 (algorytm CYK parsowania gramatyk bezkontekstowych)**.** Niech dana będzie gramatyka bezkontekstowa $G = (V, T, R, S)$ oraz łańcuch $w = w_1 \dots w_n$. Dodatkowo, niech $m = |V|$ oznacza liczbę symboli nieterminalnych w gramatyce G.

1. Utwórz tablicę t o wymiarach $n \times n \times m$ i wypełnij ją zerami.
2. Dla i od 1 do n wykonaj:
   — dla wszystkich symboli nieterminalnych A wykonaj:
      — jeśli istnieje produkcja $A \to w_i$, to $t_{i,i,A} := 1$ oraz dopisz do tej komórki wskaźnik oznaczający, że dotarliśmy do liścia drzewa wyprowadzenia.
3. Dla j od 2 do n wykonaj:
   — dla i od 1 do $n - j + 1$ wykonaj:
      — dla k od 1 do $j - 1$ wykonaj:
         — dla wszystkich trójek symboli nieterminalnych (A, B, C) wykonaj:
            — jeśli $t_{i,k,B} = 1$, $t_{i+k,j-k,C} = 1$ oraz istnieje produkcja $A \to BC$, to $t_{i,j,A} := 1$ oraz dopisz do tej komórki wskaźnik do komórek $t_{i,k,B}$ i $t_{i+k,j-k,C}$.
4. Jeżeli w komórce $t_{1,n,S}$ znajduje się 1, oznacza to, że łańcuch w należy do języka $L(G)$ generowanego przez gramatykę G. Podążając za wskaźnikami, można odczytać drzewo wyprowadzenia łańcucha w (jedno z możliwych).

Twierdzenie 3.6. *Złożoność czasowa algorytmu CYK dla łańcucha wejściowego o długości n w najgorszym przypadku wynosi $O(n^3)$.*

Dowód. Najbardziej znaczący w algorytmie jest krok 3, który wykonywany jest w czasie $O(n^3)$ (jako trzykrotnie zagnieżdżoną pętla, która przebiega po kolejnych symbolach wejściowego łańcucha) — przy założeniu, że liczbę symboli pomocniczych gramatyki traktujemy jako stałą. Z tego powodu złożoność algorytmu CYK w najgorszym przypadku wynosi właśnie $O(n^3)$. □

[7] Na podstawie [7].

3.3.2. Algorytm CYK dla probabilistycznych gramatyk bezkontekstowych

Algorytm CYK dla probabilistycznych gramatyk bezkontekstowych służy do znajdowania drzewa Viterbiego dla danego łańcucha symboli terminalnych. Podobnie jak w przypadku zwykłego algorytmu CYK, używamy trójwymiarowej tablicy o wymiarach $n \times n \times m$, gdzie n jest długością łańcucha, a m — liczbą symboli nieterminalnych w gramatyce. W każdej komórce $t_{i,j,A}$ tablicy t przechowywane jest maksymalne prawdopodobieństwo wyprowadzenia łańcucha $w = w_i \dots w_j$ z symbolu A oraz wskaźnik do kolejnego elementu drzewa wyprowadzenia.

Algorytm 3.12 (algorytm CYK parsowania probabilistycznych gramatyk bezkontekstowych)**.** Niech dana będzie probabilistyczna gramatyka bezkontekstowa $G = (V, T, R, S, P)$ oraz łańcuch $w = w_1 \dots w_n$. Dodatkowo, niech $m = |V|$ oznacza liczbę symboli nieterminalnych w gramatyce G.

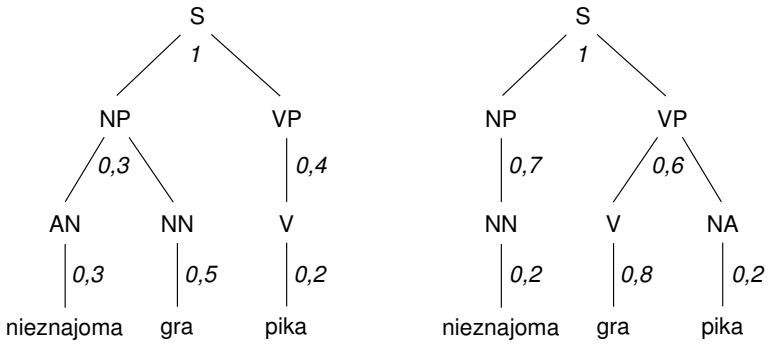
1. Utwórz tablicę t o wymiarach $n \times n \times m$ i zainicjalizuj ją zerami.
2. Dla i od 1 do n wykonaj:
   — dla wszystkich symboli nieterminalnych A wykonaj:
      — jeśli istnieje produkcja $A \to w_i$, to $t_{i,i,A} := P(A \to w_i)$ raz dopisz do tej komórki wskaźnik oznaczający, że dotarliśmy do liścia drzewa wyprowadzenia.
3. Dla j od 2 do n wykonaj:
   — dla i od 1 do $n - j + 1$ wykonaj:
      — dla k od 1 do $j - 1$ wykonaj:
         — dla wszystkich trójek symboli nieterminalnych (A, B, C) wykonaj:
            a) $p := t_{i,k,B} \cdot t_{i+k,j-k,C} \cdot P(A \to BC)$;
            b) jeśli $p > t_{i,j,A}$, to $t_{i,j,A} := p$ oraz zapisz w tej komórce wskaźniki do komórek $t_{i,k,B}$ i $t_{i+k,j-k,C}$.
4. W komórce $t_{1,n,S}$ znajduje się prawdopodobieństwo wyprowadzenia Viterbiego dla łańcucha w. Wyprowadzenie to można odczytać, podążając za wskaźnikami.

Przykład 3.1 (algorytm CYK parsowania probabilistycznych gramatyk bezkontekstowych)**.** Niech dana będzie probabilistyczna gramatyka bezkontekstowa w postaci normalnej Chomsky'ego z przykładu 2.27 na stronie 23.

Niech wejściowym łańcuchem będzie zdanie *nieznajoma gra pika* . To zdanie jest dobre dla naszego przykładu, ponieważ posiada dwa różne drzewa wyprowadzenia (rys. 3.1).

Na początku tworzymy pustą trójwymiarową tablicę t o wymiarach $3 \times 3 \times 7$:

	1 (S,NN,NA,NP,AN,V,VP)	2 (S,NN,NA,NP,AN,V,VP)	3 (S,NN,NA,NP,AN,V,VP)
1	$(0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0)$
wskaźniki	(—,—,—,—,—,—,—)	(—,—,—,—,—,—,—)	(—,—,—,—,—,—,—)
2	$(0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0)$
wskaźniki	(—,—,—,—,—,—,—)	(—,—,—,—,—,—,—)	(—,—,—,—,—,—,—)
3	$(0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0)$
wskaźniki	(—,—,—,—,—,—,—)	(—,—,—,—,—,—,—)	(—,—,—,—,—,—,—)
	nieznajoma	*gra*	*pika*



Rysunek 3.1. Dwa różne drzewa wyprowadzenia łańcucha *nieznajoma gra pika* . Drzewo po lewej ma prawdopodobieństwo 0.0036. Drzewo po prawej ma prawdopodobieństwo 0.01344. Drzewo po prawej jest zatem drzewem Viterbiego tego łańcucha.

Teraz znajdujemy reguły, których następniki są symbolami końcowymi występującymi w łańcuchu wejściowym:

$$
\begin{aligned}
&P(\, NN \;\rightarrow\; \textit{nieznajoma}\,) = 0.2,\\
&P(\, AN \;\rightarrow\; \textit{nieznajoma}\,) = 0.3,\\
&P(\, NN \;\rightarrow\; \textit{gra}\,) = 0.5,\\
&P(\, V \;\rightarrow\; \textit{gra}\,) = 0.8,\\
&P(\, NN \;\rightarrow\; \textit{pika}\,) = 0.3,\\
&P(\, NA \;\rightarrow\; \textit{pika}\,) = 0.2,\\
&P(\, V \;\rightarrow\; \textit{pika}\,) = 0.2.
\end{aligned}
$$

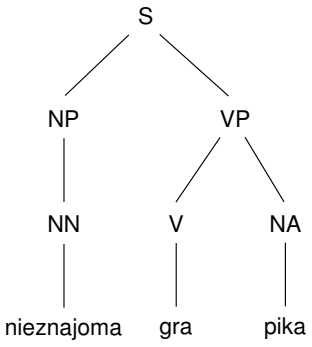
Prawdopodobieństwa tych reguł wpisujemy w odpowiednie miejsca tablicy:

	1 (S,**NN**,NA,NP,**AN**,V,VP)	2 (S,**NN**,NA,NP,AN,**V**,VP)	3 (S,**NN**,**NA**,NP,AN,**V**,VP)
1	$(0, \mathbf{0.2}, 0, 0, \mathbf{0.3}, 0, 0)$	$(0, 0, 0, 0, 0, 0, 0)$	$(0, 0, 0, 0, 0, 0, 0)$
wsk.	(—,/,—,—,/,—,—)	(—,—,—,—,—,—,—)	(—,—,—,—,—,—,—)
2	$(0, 0, 0, 0, 0, 0, 0)$	$(0, \mathbf{0.5}, 0, 0, 0, \mathbf{0.8}, 0)$	$(0, 0, 0, 0, 0, 0, 0)$
wsk.	(—,—,—,—,—,—,—)	(—,/,—,—,—,/,—)	(—,—,—,—,—,—,—)
3	$(0, 0, 0, 0, 0, 0, 0)$	$(0, 0, 0, 0, 0, 0, 0)$	$(0, \mathbf{0.3}, \mathbf{0.2}, 0, 0, \mathbf{0.2}, 0)$
wsk.	(—,—,—,—,—,—,—)	(—,—,—,—,—,—,—)	(—,/,/,—,—,/,—)
	nieznajoma	*gra*	*pika*

Kolejnym krokiem jest znalezienie symboli, które rozwijają się w podłańcuchy długości 2:

$$
\begin{aligned}
&NP \;\rightarrow\; AN\ NN\,,\\
&\qquad p := 0.3 \cdot 0.5 \cdot P(\, NP \;\rightarrow\; AN\ NN\,) = 0.3 \cdot 0.5 \cdot 1 = 0.15;\\
&VP \;\rightarrow\; V\ NA\,,\\
&\qquad p := 0.8 \cdot 0.2 \cdot P(\, VP \;\rightarrow\; V\ NA\,) = 0.8 \cdot 0.2 \cdot 1 = 0.16.
\end{aligned}
$$

Znaleźliśmy dwa takie symbole. Aktualizujemy wartości w tablicy i przypisujemy odpowiednie wskaźniki:



Rysunek 3.2. Drzewo Viterbiego uzyskane za pomocą algorytmu CYK.

	1 (S,NN,NA,NP,*AN*,V,VP)	2 (S,*NN*,NA,**NP**,AN,*V*,VP)	3 (S,NN,*NA*,NP,AN,V,**VP**)
1	$(0, 0.2, 0, 0, \mathit{0.3}, 0, 0)$	$(0, 0, 0, \mathbf{0.15}, 0, 0, 0)$	$(0, 0, 0, 0, 0, 0, 0)$
wsk.	(—,/,—,—,/,—,—)	(–,–,–,**[1,1,AN;2,2,NN]**,–,–,–)	(—,—,—,—,—,—,—)
2	$(0, 0, 0, 0, 0, 0, 0)$	$(0, \mathit{0.5}, 0, 0, 0, \mathit{0.8}, 0)$	$(0, 0, 0, 0, 0, 0, \mathbf{0.16})$
wsk.	(—,—,—,—,—,—,—)	(—,/,—,—,—,/,—)	(–,–,–,–,–,–,**[2,2,V;3,3,NA]**)
3	$(0, 0, 0, 0, 0, 0, 0)$	$(0, 0, 0, 0, 0, 0, 0)$	$(0, 0.3, \mathit{0.2}, 0, 0, 0.2, 0)$
wsk.	(—,—,—,—,—,—,—)	(—,—,—,—,—,—,—)	(—,/,/,—,—,/,—)
	nieznajoma	*gra*	*pika*

W ostatnim kroku znajdujemy symbole, które rozwijają się w podłańcuchy długości 3, czyli w naszym przypadku — w cały łańcuch. Znajdujemy dwie pasujące reguły, których poprzednikiem jest symbol S:

$$S \rightarrow NP\ V,$$
$$p_1 := 0.15 \cdot 0.2 \cdot P(S \rightarrow NP\ V) = 0.15 \cdot 0.2 \cdot 0.12 = 0.0036;$$
$$S \rightarrow NN\ VP,$$
$$p_2 := 0.2 \cdot 0.16 \cdot P(S \rightarrow NN\ VP) = 0.2 \cdot 0.16 \cdot 0.42 = 0.01344.$$

Ponieważ $p_2 > p_1$, zatem w komórce $t_{1,3,S}$ znajdzie się wartość p_2 oraz wskaźnik na poddrzewa o korzeniach w NN i VP:

	1 (S,*NN*,NA,NP,AN,V,VP)	2 (S,NN,NA,NP,AN,V,VP)	3 (**S**,NN,NA,NP,AN,V,*VP*)
1	$(0, \mathit{0.2}, 0, 0, 0.3, 0, 0)$	$(0, 0, 0, 0.15, 0, 0, 0)$	$(\mathbf{0.01344}, 0, 0, 0, 0, 0, 0)$
wsk.	(—,/,—,—,/,—,—)	(–,–,–,[1,1,AN;2,2,NN],–,–,–)	(**[1,1,NN;2,3,VP]**,–,–,–,–,–,–)
2	$(0, 0, 0, 0, 0, 0, 0)$	$(0, 0.5, 0, 0, 0, 0.8, 0)$	$(0, 0, 0, 0, 0, 0, \mathit{0.16})$
wsk.	(—,—,—,—,—,—,—)	(—,/,—,—,—,/,—)	(–,–,–,–,–,–,[2,2,V;3,3,NA])
3	$(0, 0, 0, 0, 0, 0, 0)$	$(0, 0, 0, 0, 0, 0, 0)$	$(0, 0.3, 0.2, 0, 0, 0.2, 0)$
wsk.	(—,—,—,—,—,—,—)	(—,—,—,—,—,—,—)	(—,/,/,—,—,/,—)
	nieznajoma	*gra*	*pika*

Teraz możemy odczytać drzewo Viterbiego podążając za wskaźnikami z komórki $t_{1,3,S}$ (rys. 3.2).

Podobnie jak dla wersji nieprobabilistycznej (i z tych samych powodów), złożoność czasowa probabilistycznego algorytmu CYK wynosi w najgorszym przypadku $O(n^3)$ dla łańcucha wejściowego o długości n.

3.4. Algorytmy parsowania wykorzystujące metody z algorytmów grafowych

3.4.1. Wykorzystanie algorytmów grafowych do parsowania

Do parsowania gramatyk bezkontekstowych można wykorzystać narzędzia teorii grafów. Jeżeli skonstruuje się odpowiedni graf na bazie danej gramatyki, można zastąpić zagadnienie szukania drzew wyprowadzeń łańcucha przez zagadnienie znajdowania dróg między określonymi wierzchołkami tego grafu.

Można zauważyć, że podstawowe metody parsowania — podejście zstępujące i podejście wstępujące — polegają w gruncie rzeczy na szukaniu sposobów połączenia symbolu początkowego gramatyki z parsowanym łańcuchem za pomocą wyprowadzeń. Jeżeli potraktuje się łańcuchy jako wierzchołki grafu, stosowane do nich reguły produkcji jako krawędzie, prawdopodobieństwa reguł jako wagi, a drzewa wyprowadzenia jako ścieżki, to można przeformułować problem parsowania łańcucha na zagadnienie znajdowania ścieżki łączącej wierzchołek odpowiadający temu łańcuchowi z wierzchołkiem odpowiadającym symbolowi początkowemu.

Definicja 3.1 (graf gramatyki)**.** Niech $G = (V, T, R, S, P)$ będzie probabilistyczną gramatyką bezkontekstową. Wówczas multigraf skierowany z wagami $\mathbf{\Gamma} = (U, E, \gamma, W)$ będziemy nazywać *grafem gramatyki* G, jeżeli spełnione są następujące warunki:

(1) zbiór wierzchołków $U = \{\omega \in (V \cup T)^+ \colon S \Rightarrow_G^* \omega\}$,
(2) zbiór krawędzi $E \subseteq U \times U \times R$,
(3) jeżeli $(\zeta, \xi, r) \in E$, to łańcuch ξ jest bezpośrednio wyprowadzalny z łańcucha ζ przy użyciu reguły r,
(4) jeżeli $\zeta, \xi \in U$ oraz $\zeta \Rightarrow_G \xi$, to dla każdej reguły r, której zastosowanie do łańcucha ζ produkuje łańcuch ξ, istnieje dokładnie jedna krawędź $e = (\zeta, \xi, r) \in E$,
(5) każda krawędź $e = (\zeta, \xi, r) \in E$ biegnie od wierzchołka ζ do wierzchołka ξ, tj.:
$$\gamma(e) = \gamma((\zeta, \xi, r)) = (\zeta, \xi),$$
(6) dla każdej krawędzi $e = (\zeta, \xi, r) \in E$ jej waga równa jest ujemnemu logarytmowi[8] prawdopodobieństwa odpowiadającej jej reguły:
$$W(e) = W((\zeta, \xi, r)) = -\log P(r).$$

Graf każdej probabilistycznej gramatyki bezkontekstowej jest acyklicznym multigrafem skierowanym, ponieważ każda gramatyka bezkontekstowa jest monotoniczna.

Ponieważ prawdopodobieństwa reguł są liczbami rzeczywistymi z przedziału $[0, 1]$, zatem wagi krawędzi grafu gramatyki mieszczą się w przedziale $[0, \infty]$, przy czym waga krawędzi jest równa ∞ tylko wtedy, gdy prawdopodobieństwo odpowiadającej jej reguły wynosi 0. W praktyce oznacza to, że reguła ta nigdy nie zostanie użyta, zatem równie dobrze można usunąć tę regułę ze zbioru reguł R gramatyki, a odpowiadające jej krawędzie usunąć z grafu. Można zatem

[8] W niniejszej pracy będę używał logarytmów o podstawie 2, chyba że będzie wyraźnie zaznaczone, że jest inaczej.

przyjąć, że wszystkie krawędzie grafu gramatyki posiadają skończone nieujemne wagi.

Każdej ścieżce w grafie gramatyki odpowiada dokładnie jedno wyprowadzenie wraz z ciągiem produkcji użytych do wygenerowania kolejnych łańcuchów tego wyprowadzenia. Istotnie, ścieżka taka jest ciągiem sąsiednich wierzchołków, a te są łańcuchami bezpośrednio wyprowadzalnymi jeden z drugiego przy użyciu reguł odpowiadającym łączącym je krawędziom.

Suma wag krawędzi ścieżki łączącej symbol początkowy gramatyki $G = (V, T, R, S, P)$ z łańcuchem $\omega \in (V \cup T)^+$ jest równa

$$W(e_1) + \cdots + W(e_k) = (-\log P(r_1)) + \cdots + (-\log P(r_k)) = \\ = -\log(P(r_1) \cdots P(r_k)),$$

czyli ujemnemu logarytmowi prawdopodobieństwa odpowiadającego jej wyprowadzenia ($e_1, \ldots, e_k$ oznaczają krawędzie ścieżki, $r_1, \ldots, r_k$ oznaczają reguły wyprowadzenia). Zauważmy, że jednemu drzewu wyprowadzenia może odpowiadać wiele ścieżek w grafie gramatyki, ponieważ jednemu drzewu wyprowadzenia może odpowiadać wiele wyprowadzeń. Nie zmienia to jednak faktu, że ścieżka o najmniejszej sumie wag krawędzi odpowiada wyprowadzeniu o największym prawdopodobieństwie, a zatem drzewu Viterbiego łańcucha ω.

Oznacza to, że do znajdowania drzewa Viterbiego można użyć algorytmów grafowych służących do odnajdywania najkrótszej ścieżki (ścieżki o najmniejszej wadze) między dwoma wierzchołkami grafu.

3.4.2. Strategia przeszukiwania „najpierw najlepszy”[9]

Przeszukiwanie „najpierw najlepszy” (ang. *best-first search*) jest algorytmem przeszukiwania grafu polegającym na tym, że najpierw odwiedza się najbardziej obiecujące wierzchołki (czy też krawędzie). Dana jest przy tym *funkcja oceny* (nazywana też *miarą wartości* — ang. *figure of merit*), która określa dla każdego wierzchołka, jak bardzo wart jest on odwiedzenia.

Algorytm parsowania „najpierw najlepszy” jest rozwinięciem algorytmu parsowania tablicowego. O kolejności wyjmowania składników z agendy decyduje funkcja oceny. Funkcje oceny używane w parsingu probabilistycznych gramatyk bezkontekstowych są oparte na prawdopodobieństwach reguł. Wykorzystuje się w tym celu takie miary, jak prawdopodobieństwo wewnętrzne składnika (podłańcucha), unormowane prawdopodobieństwo wewnętrzne czy oszacowania trigramowe.

Zaletą podejścia „najpierw najlepszy” jest istotne zmniejszenie nakładu pracy wykonywanej przez algorytm, dzięki temu, że składniki, których miara wartości nie jest wysoka, nie są przetwarzane. Wadą jest to, że algorytm ten nie gwarantuje, że znalezione przezeń drzewo rozkładu jest drzewem Viterbiego. Również asymptotyczna złożoność czasowa w najgorszym przypadku pozostaje sześcienna ($O(n^3)$ dla łańcucha wejściowego o długości n), a przy niektórych miarach wartości może być nawet wyższa ($O(n^5)$).

[9] Na podstawie [1].

3.4.3. Strategia przeszukiwania wiązkowego[10]

Strategia przeszukiwania wiązkowego (ang. *beam search*) jest rozszerzeniem metody „najpierw najlepszy". Polega ona na tym, że ogranicza się z góry liczbę ścieżek, które są brane pod uwagę jako kandydatki na źródła najbardziej obiecujących wierzchołków. To ograniczenie jest właśnie ową „szerokością wiązki", od której metoda wzięła swoją nazwę.

W przypadku algorytmów parsingu tablicowego przeszukiwanie wiązkowe realizowane jest w ten sposób, że w agendzie trzymana jest ograniczona liczba elementów. Podczas dodawania nowo skonstruowanych elementów do agendy, elementy o najmniejszej wartości funkcji oceny są odrzucane. Oznacza to, że w danym momencie śledzona jest ograniczona liczba możliwych drzew rozkładu wejściowego łańcucha.

Złożoność czasowa tego algorytmu jest liniowa i może być dowolnie poprawiona przez odpowiednie zmniejszenie szerokości wiązki[8].

W kolejnych krokach algorytmu część potencjalnych rozwiązań, które nie są lokalnie optymalne, jest odrzucana. Ponieważ globalnie optymalne drzewo Viterbiego nie musi być lokalnie najlepsze, jest możliwe, że zostanie odrzucone w trakcie działania algorytmu. Istnieje zatem ryzyko, że otrzymane na końcu drzewo rozkładu wejściowego łańcucha nie będzie drzewem Viterbiego, co jest wadą algorytmu parsowania metodą przeszukiwania wiązkowego.

[10] Na podstawie [11] i [12].

Rozdział 4

Algorytm A* w parsingu — opis i porównanie różnych rodzajów heurystyk

4.1. Algorytm A* — opis działania w ogólnym przypadku

Algorytm A* jest jednym z algorytmów służących do znajdowania najkrótszej ścieżki między dwoma danymi wierzchołkami grafu (grafu skierowanego, multigrafu skierowanego) z wagami. Jest odmianą algorytmu „najpierw najlepszy".

Przyjmijmy, że chcemy znaleźć najkrótszą ścieżkę z wierzchołka u do wierzchołka v w grafie (grafie skierowanym, multigrafie skierowanym) ważonym Γ, którego zbiorem wierzchołków jest U, zbiorem krawędzi — E, a funkcją wagi — W.

W celu znalezienia najkrótszej ścieżki do wierzchołka v algorytm A* wykorzystuje dodatkową funkcję $h\colon U \to \mathbb{R}$, nazywaną *heurystyką* albo *funkcją oceny*. Heurystyka $h(x)$ określa, jaka jest przewidywana odległość od wierzchołka x do wierzchołka v[1]. Na ogół wartość funkcji h nie jest dokładnie równa rzeczywistej długości najkrótszej ścieżki z x do v, lecz jedynie ją przybliża, ale za to przyjmuje się, że jest dana z góry lub łatwa do obliczenia. Rodzaj użytej heurystyki ma wpływ na właściwości algorytmu. Jeśli nałożymy odpowiednie ograniczenia na stosowaną heurystykę, możemy spodziewać się, że uzyskamy algorytm o żądanych własnościach.

Algorytm rozpoczyna działanie od wierzchołka początkowego u i próbuje budować drogę w ten sposób, aby zmaksymalizowana została wartość funkcji $f(x) = g(x) + h(x)$, gdzie dla danego wierzchołka $x \in U$ wartość $g(x)$ jest obliczoną już długością ścieżki z u do x. Ściślej mówiąc, $g(x)$ jest sumą wag krawędzi należących do już zbudowanej ścieżki powiększoną o wagę krawędzi łączącej aktualnie przetwarzany wierzchołek z wierzchołkiem x.

Algorytm wykorzystuje dwa zbiory wierzchołków:
— w zbiorze C znajdują się wierzchołki, które już zostały przetworzone,
— zbiór D przechowuje jeszcze nie odwiedzone wierzchołki.
W trakcie działania algorytmu wybierany jest ten wierzchołek x_0 ze zbioru D, dla którego wartość funkcji f jest najmniejsza. Następnie przeglądane są nieprzetworzone jeszcze wierzchołki y sąsiadujące z wierzchołkiem x_0. Obliczana jest tymczasowa wartość g', która jest równa sumie wartości funkcji g dla wierzchołka x_0 oraz odległości między przeglądanym właśnie wierzchołkiem y a wierzchołkiem x_0. Jeżeli wierzchołek y nie był wcześniej odwiedzany lub jeśli obliczona wartość g' jest mniejsza od dotychczasowej wartości $g(y)$, to wartość g' przyjmowana jest jako nowa wartość funkcji $g(y)$, a wierzchołkowi y przypisywany jest

[1] Przypomnijmy, że odległość $d(u, v)$ między wierzchołkami $u, v \in U$ grafu $\Gamma = (U, E)$ definiujemy jako długość najkrótszej ścieżki z u do v, czyli sumę wag krawędzi składających się na tę ścieżkę — patrz definicje 2.23 i 2.24 na stronie 11.

wskaźnik do wierzchołka x_0. Aktualizowana jest też odpowiednio wartość funkcji $f(y)$. Przypisane wierzchołkom wskaźniki umożliwiają odtworzenie szukanej ścieżki po zakończeniu algorytmu.

Poniższy algorytm znajduje najkrótszą ścieżkę między dwoma danymi wierzchołkami grafu lub grafu skierowanego. Działa również dla multigrafów skierowanych, z tym, że należy uwzględnić, iż wierzchołek x może być wówczas połączony z y więcej niż jedną krawędzią. Wówczas wybieramy z nich tę o najmniejszej wadze, a resztę odrzucamy.

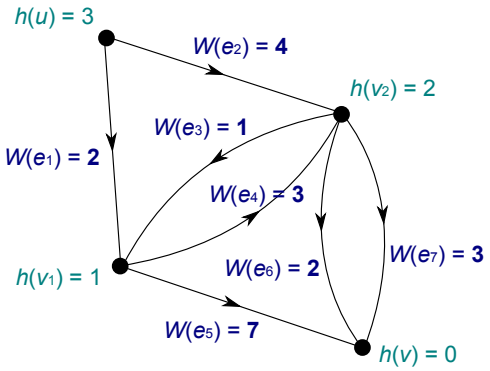
Algorytm 4.1 (A*)**.** Niech dany będzie graf (graf skierowany, multigraf skierowany) Γ o zbiorze wierzchołków U, zbiorze krawędzi E i funkcji wagi W. Niech dane będą wierzchołki u i v, między którymi chcemy znaleźć najkrótszą drogę oraz funkcja heurystyki $h\colon U \to \mathbb{R}$.

1. Utwórz pusty zbiór C.
2. Utwórz pusty zbiór D.
3. Dodaj wierzchołek u do zbioru D.
4. $g(u) := 0$.
5. $f(u) := g(u) + h(u) = 0 + h(u) = h(u)$.
6. Dopóki zbiór D nie jest pusty, wykonuj:
   a) znajdź w zbiorze D wierzchołek, dla którego wartość funkcji f jest najmniejsza:
   $$x_0 := \arg\min_{x \in D} f(x);$$
   b) jeżeli $x_0 = v$, to zakończ działanie algorytmu i jako wynik zwróć znalezioną ścieżkę (podążając za wskaźnikami);
   c) usuń wierzchołek x_0 ze zbioru D;
   d) dodaj wierzchołek x_0 do zbioru C;
   e) dla każdego wierzchołka $y \notin C$ sąsiadującego z x_0 wykonuj:
      i. $g' := g(x_0) + W(e)$, gdzie $e \in E$ jest (najkrótszą — w przypadku multigrafu) krawędzią łączącą wierzchołek x_0 z wierzchołkiem y,
      ii. jeśli $y \notin D$, to:
         A. dodaj y do zbioru D,
         B. wskaźnik$(y) := x_0$,
         C. $g(y) := g'$,
         D. $f(y) := g(y) + h(y)$,
      iii. jeśli $y \in D$ oraz $g' < g(y)$, to:
         A. wskaźnik$(y) := x_0$,
         B. $g(y) := g'$,
         C. $f(y) := g(y) + h(y)$.
7. Jeżeli algorytm nie zakończył się w kroku 6b, to zwróć informację, że algorytm nie znalazł ścieżki z u do v.

Definicja 4.1 (heurystyka dopuszczalna)**.** Mówimy, że heurystyka h jest *dopuszczalna*, jeżeli dla dowolnych dwóch wierzchołków x i y spełniona jest nierówność

$$h(x) \leqslant d(x, y) + h(y),$$

gdzie $d(x, y)$ oznacza odległość między wierzchołkami x i y, czyli długość najkrótszej ścieżki z x do y.



Rysunek 4.1. Multigraf skierowany z wagami przed rozpoczęciem działania algorytmu A*.

Twierdzenie 4.1. *Jeżeli heurystyka jest dopuszczalna, to algorytm A* zawsze znajdzie najkrótszą ścieżkę między danymi dwoma wierzchołkami grafu (grafu skierowanego, multigrafu skierowanego).*

Dowód. Przyjmijmy, że algorytm szuka najkrótszej ścieżki z wierzchołka u do wierzchołka v.

Niech $(y_0, y_1, \ldots, y_k)$, $y_0 = x$, $y_k = v$, będzie ścieżką z wierzchołka x do wierzchołka v. Ponieważ heurystyka jest dopuszczalna, zatem zachodzą nierówności:

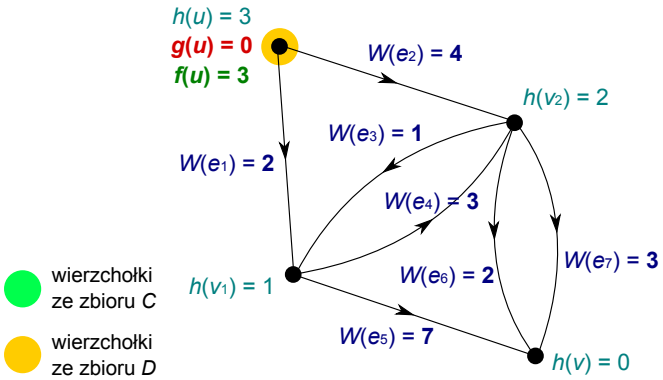
$$h(x) = h(y_0) \leqslant d(y_0, y_1) + h(y_1) \leqslant d(y_0, y_1) + d(y_1, y_2) + h(y_2) \leqslant$$
$$\leqslant d(y_0, y_1) + \cdots + d(y_{k-1}, y_k) = \sum_{i=1}^{k} d(y_{i-1}, y_i).$$

Oznacza to, że wartość heurystyki $h(x)$ jest nie większa od rzeczywistej długości każdej ścieżki z x do v. Wynika stąd, że wartość $f(x) = g(x) + h(x)$ jest nie większa od rzeczywistej długości najkrótszej ścieżki z u do v przechodzącej przez wierzchołek x. Ponieważ algorytm na każdym kroku stara się zminimalizować wartość $f(x)$, zatem jeżeli napotka wierzchołek znajdujący się na najkrótszej ścieżce z u do v, to wierzchołek ten zostanie wybrany. □

Przykład 4.1 (algorytm A*)**.** Niech dany będzie multigraf skierowany z wagami z przykładu 2.4 na stronie 11. Oznaczmy jego wierzchołki przez u, v, v_1 i v_2, zaś krawędzie przez $e_1, e_2, \ldots, e_7$. Przyjmijmy, że chcemy znaleźć najkrótszą drogę z wierzchołka u do wierzchołka v. Niech funkcja heurystyki będzie dana w następujący sposób:

$$h(u) = 3,$$
$$h(v) = 0,$$
$$h(v_1) = 1,$$
$$h(v_2) = 2.$$

Ta sytuacja początkowa przedstawiona jest na rysunku 4.1.



Rysunek 4.2. Sytuacja po wykonaniu pierwszego kroku algorytmu A*.

Niech $d(x, y)$ oznacza odległość od wierzchołka x do wierzchołka y. Zauważmy, że funkcja heurystyki jest dopuszczalna. Istotnie[2]:

$$h(u) = 3 \leqslant 6 = 6 + 0 = d(u, v) + h(v), \quad d(u, v) = W(e_2) + W(e_6) = 6,$$
$$h(u) = 3 \leqslant 3 = 2 + 1 = d(u, v_1) + h(v_1), \quad d(u, v_1) = W(e_1) = 2,$$
$$h(u) = 3 \leqslant 6 = 4 + 2 = d(u, v_2) + h(v_2), \quad d(u, v_2) = W(e_2) = 4,$$
$$h(v_1) = 1 \leqslant 5 = 5 + 0 = d(v_1, v) + h(v), \quad d(v_1, v) = W(e_4) + W(e_6) = 5,$$
$$h(v_1) = 1 \leqslant 5 = 3 + 2 = d(v_1, v_2) + h(v_2), \quad d(v_1, v_2) = W(e_4) = 3,$$
$$h(v_2) = 2 \leqslant 2 = 2 + 0 = d(v_2, v) + h(v), \quad d(v_2, v) = W(e_6) = 2,$$
$$h(v_2) = 2 \leqslant 2 = 1 + 1 = d(v_2, v_1) + h(v_1), \quad d(v_2, v_1) = W(e_3) = 1.$$

Dalsza część przykładu przedstawia działanie algorytmu A* znajdującego najkrótszą drogę od wierzchołka u do wierzchołka v w tym multigrafie.

Na początku wierzchołek startowy u dodawany jest do zbioru D, przypisywana jest mu wartość $g(u) := 0$ i $f(u) := h(u) = 3$ (rys. 4.2).

Zbiór D zawiera teraz tylko jeden wierzchołek, mianowicie u, dlatego wybieramy właśnie ten wierzchołek. Przenosimy wierzchołek u ze zbioru D do zbioru C i przeglądamy jego nieprzetworzonych sąsiadów.

Pierwszym nieprzetworzonym sąsiadem jest $v_1 \notin D$. Wykonujemy odpowiednie obliczenia:

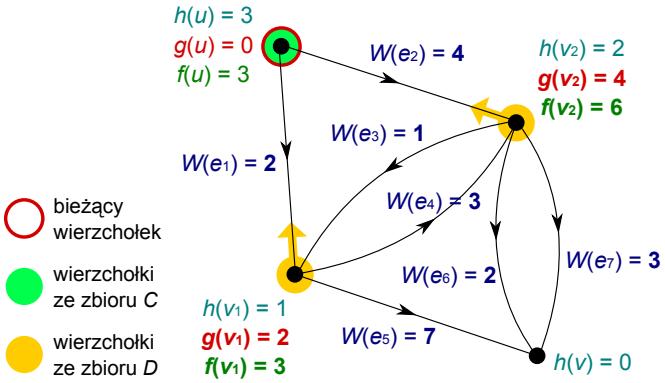
$$g' := g(u) + W(e_1) = 0 + 2 = 2,$$
$$g(v_1) := g' = 2,$$
$$f(v_1) := g(v_1) + h(v_1) = 2 + 1 = 3.$$

Podobnie postępujemy dla wierzchołka $v_2 \notin D$:

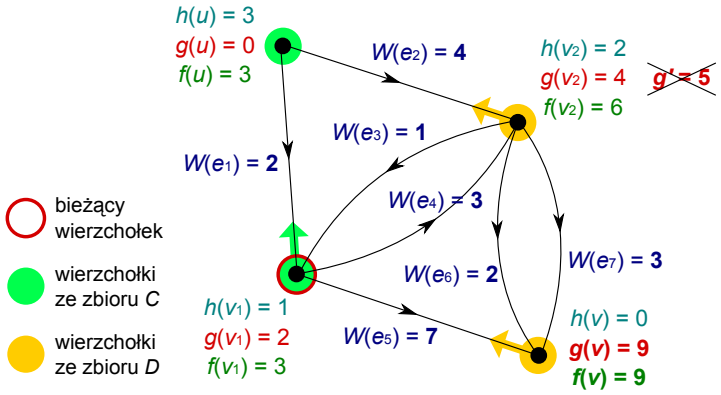
$$g' := g(u) + W(e_2) = 0 + 4 = 4,$$
$$g(v_2) := g' = 4,$$
$$f(v_2) := g(v_2) + h(v_2) = 4 + 2 = 6.$$

Ustawiamy również odpowiednio wskaźniki. Sytuacja po tym kroku pokazana jest na rysunku 4.3.

[2]Zwróćmy uwagę, że odległość z wierzchołka v_1 do wierzchołka v wynosi 5 (suma długości krawędzi e_4 i e_6), a nie 7 (długość krawędzi e_5 łączącej v_1 i v).



Rysunek 4.3. Sytuacja po wykonaniu drugiego kroku algorytmu A*.



Rysunek 4.4. Sytuacja po wykonaniu trzeciego kroku algorytmu A*.

Teraz bierzemy ten wierzchołek ze zbioru D, który ma najmniejszą wartość funkcji f, czyli v_1.

Pierwszym nieprzetworzonym sąsiadem wierzchołka v_1 jest wierzchołek $v \notin D$:

$$\begin{aligned} g' &:= g(v_1) + W(e_5) = 2 + 7 = 9, \\ g(v) &:= g' = 9, \\ f(v) &:= g(v) + h(v) = 9 + 0 = 9. \end{aligned}$$

Drugim nieprzetworzonym sąsiadem wierzchołka v_1 jest odwiedzony już wierzchołek $v_2 \in D$. Z nim postępujemy trochę inaczej. Najpierw obliczamy wartość

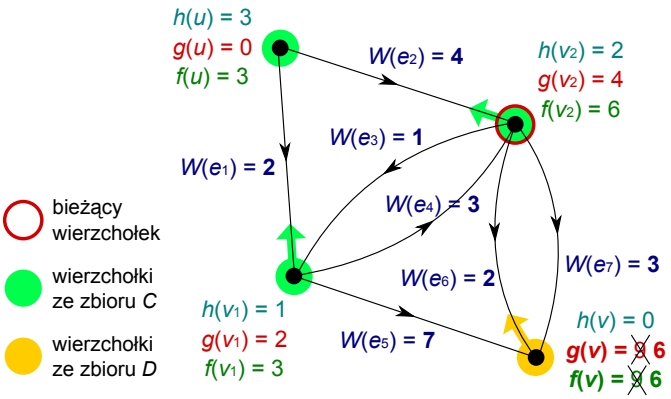
$$g' := g(v_1) + W(e_4) = 2 + 3 = 5.$$

Teraz sprawdzamy, czy nowa wartość g' jest mniejsza od $g(v_2)$:

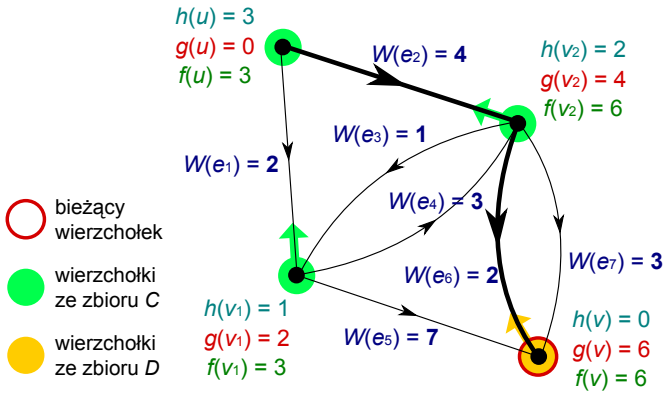
$$g' = 5 > 4 = g(v_2).$$

Okazuje się, że wartość g' nie jest mniejsza od $g(v_2)$, dlatego nie przetwarzamy dalej tego wierzchołka — przypisaną mu wartość g pozostawiamy niezmienioną.

Sytuację po tym kroku przedstawia rys. 4.4.



Rysunek 4.5. Sytuacja po wykonaniu czwartego kroku algorytmu A*.



Rysunek 4.6. Najkrótsza ścieżka z wierzchołka u do wierzchołka v znaleziona przez algorytm A*.

Teraz wierzchołkiem z D, dla którego wartość funkcji f jest najmniejsza, jest wierzchołek v_2. Jedynym nieprzetworzonym jeszcze sąsiadem tego wierzchołka jest $v \in D$. Jednak prowadzą doń z v_2 dwie krawędzie (e_6 i e_7). Wybieramy tę o mniejszej wadze, czyli e_6. Obliczamy nową wartość g' dla v:

$$g' = g(v_2) + W(e_6) = 4 + 2 = 6.$$

Wartość ta jest mniejsza niż dotychczasowa wartość g dla v:

$$g' = 6 < 9 = g(v).$$

Dlatego aktualizujemy odpowiednie wartości i wskaźnik dla wierzchołka v (rys. 4.5):

$$g(v) := g' = 6,$$
$$f(v) := g(v) + h(v) = 6 + 0 = 6.$$

Teraz w zbiorze D pozostał już tylko docelowy wierzchołek v. Pozostaje teraz już tylko odtworzyć ścieżkę podążając za wskaźnikami i zakończyć działanie algorytmu (rys. 4.6).

4.2. Algorytm A* w parsingu[3]

4.2.1. Ogólny opis działania algorytmu parsingu A*

Algorytm parsingu A* jest odmianą algorytmu tablicowego. Właściwie jedynym istotnym elementem, który odróżnia go od algorytmu 3.1 przedstawionego na stronie 28, jest kolejność wyjmowania elementów z agendy — najpierw wyjmowane są elementy o największej wartości funkcji f. Agendę A* można zaimplementować jako kolejkę priorytetową, w której priorytety elementów mogą się zmieniać w trakcie działania algorytmu.

Na podobnych zasadach opierają się algorytmy przeszukiwania „najpierw najlepszy" i przeszukiwania wiązkowego, opisane w sekcjach 3.4.2 i 3.4.3. W przeciwieństwie do nich, algorytm A* zawsze gwarantuje optymalność (znalezienie drzewa Viterbiego parsowanego łańcucha), pozostając jednocześnie algorytmem szybkim.

Złożoność czasowa znalezienia jakiegokolwiek poprawnego drzewa wyprowadzenia wejściowego łańcucha o długości n przez parser tablicowy wynosi $O(n^3)$. W przypadku znajdowania drzewa Viterbiego zagadnienie jest bardziej skomplikowane, ponieważ może się zdarzyć, że po wyjęciu danej krawędzi z agendy i wbudowaniu jej w drzewo wyprowadzenia zostanie znaleziony sposób wbudowania tej krawędzi dający większe prawdopodobieństwo drzewa. Trzeba wówczas przebudowywać część drzewa wyprowadzenia (zawierającą tę krawędź). Postępowanie takie zapewnia znalezienie drzewa Viterbiego, ale w zamian za to powoduje, że złożoność czasowa takiego algorytmu jest większa niż $O(n^3)$.

Aby uniknąć opisanych powyżej sytuacji przy jednoczesnym zachowaniu optymalności algorytmu, musi być spełniony warunek, by dla każdej krawędzi $X[i,j]$ wszystkie krawędzie-wierzchołki poddrzewa o korzeniu etykietowanym $X[i,j]$ zostały wyjęte z agendy, zanim zostanie wyjęta sama krawędź $X[i,j]$. Dzięki temu można mieć pewność, że po wyjęciu danej krawędzi z agendy zostanie ona wbudowana na swoje ostateczne miejsce w drzewie wyprowadzenia.

Jeżeli gramatyka jest w postaci normalnej Chomsky'ego, jednym ze sposobów, aby to osiągnąć, jest zapewnienie pierwszeństwa krawędziom krótszym (tj. o krótszych zakresach) przed dłuższymi. Powstały w ten sposób algorytm jest w istocie algorytmem CYK.

Innym sposobem jest użycie odpowiednich oszacowań, tak jak w przypadku algorytmu A*.

Niech $G = (V, T, R, S, P)$ będzie probabilistyczną gramatyką bezkontekstową, zaś $w = w_1 \dots w_n \in T^+$ łańcuchem wejściowym.

Wewnętrznym drzewem wyprowadzenia krawędzi $e = X[i,j]$ nazwiemy drzewo wyprowadzenia łańcucha $w_{i+1} \dots w_j$ z symbolu X w gramatyce G. Oznaczmy przez $\beta_G(e)$ (lub po prostu $\beta(e)$) logarytm z maksymalnego prawdopodobieństwa takiego drzewa i nazwijmy tę liczbę *wewnętrzną wagą Viterbiego* krawędzi e. Będziemy szacować wartości $\beta(e)$ za pomocą oszacowań $b(e)$, które na początku będą wynosiły $-\infty$ dla każdej krawędzi, następnie rosły, a przy wyjmowaniu krawędzi z agendy osiągną wartość $\beta(e)$.

Użycie oszacowania b jako priorytetu wyjmowania krawędzi z agendy owocuje uzyskaniem parsera, który znajduje zawsze drzewo Viterbiego. Algorytm ten

[3] Na podstawie [8].

jest odpowiednikiem algorytmu równomiernego kosztu (*uniform cost search*) stosowanego do znajdywania najkrótszej drogi w grafie.

Zewnętrznym drzewem wyprowadzenia krawędzi $e = X[i, j]$ nazwiemy drzewo wyprowadzenia łańcucha $w_1 \dots w_i X w_{j+1} \dots w_n$ z symbolu początkowego S w gramatyce G. Przez $\alpha_G(e)$ (lub po prostu $\alpha(e)$) oznaczymy logarytm z maksymalnego prawdopodobieństwa takiego drzewa i nazwijmy tę liczbę *zewnętrzną wagą Viterbiego* krawędzi e. Podobnie jak w przypadku wagi wewnętrznej, będziemy szacować wartości $\alpha(e)$ za pomocą oszacowań $a(e)$.

W algorytmie parsingu A* oszacowanie b jest odpowiednikiem funkcji g z klasycznego algorytmu A*, zaś oszacowanie a odpowiada funkcji heurystyki h. Będziemy używać wartości $b + a$ jako priorytetu wyjmowania krawędzi z agendy — tak jak używaliśmy sumy $g + h$ jako priorytetu analizowania kolejnych łuków najkrótszej drogi w klasycznym algorytmie A*.

Zauważmy, że wartość $b + a$ będzie maksymalizowana w trakcie działania algorytmu parsowania, nie zaś minimalizowana, jak było to w przypadku algorytmu znajdowania najkrótszej drogi. Wynika to z faktu, że używamy logarytmów prawdopodobieństw (które mieszczą się w przedziale $(-\infty, 0]$), nie zaś ujemnych logarytmów prawdopodobieństw (jak w sekcji 3.4.1). Wobec tego wartości tych oszacowań są niedodatnie i tym bliższe zeru, im większe jest prawdopodobieństwo.

Heurystyka a musi spełniać warunek dopuszczalności, to znaczy jej wartość nie może być mniejsza od logarytmu rzeczywistej zewnętrznej wagi Viterbiego krawędzi:

$$a(e) \geqslant \alpha(e).$$

Musi być też monotoniczna, tzn. taka, żeby suma $\beta + a$ nigdy się nie zwiększała.

W dalszej części rozdziału będę przyjmował dla uproszczenia, że rozważana probabilistyczna gramatyka bezkontekstowa dana jest w postaci zbinaryzowanej, co nie stanowi istotnego ograniczenia ze względu na twierdzenie 2.4.

4.2.2. Heurystyki oparte na oszacowaniach kontekstu

Aby oszacować zewnętrzną wagę Viterbiego krawędzi $\alpha(e)$, można użyć heurystyk opartych na wiedzy o sąsiednich krawędziach — czyli o kontekście krawędzi e. W tym celu będziemy obliczać dokładną wartość zewnętrznej wagi Viterbiego, ale nie dla rzeczywistego kontekstu, lecz jedynie dla jego przybliżenia — będącego pewnym ogólnym schematem, wzorcem odzwierciedlającym częściową wiedzę na temat tegoż kontekstu. Jeżeli do tego wzorca będzie pasować odpowiednia ilość kontekstów, możemy te oszacowania zapisywać w pamięci i wykorzystywać później, żeby nie było trzeba ich obliczać za każdym razem — oszczędzamy w ten sposób czas potrzebny na działanie algorytmu.

Możliwy jest cały zakres oszacowań, w zależności od wiedzy o kontekście, jaką będziemy wykorzystywać.

Oszacowanie oparte na zerowej wiedzy — oszacowanie trywialne NULL — jest stale równe zero. Wynika to z faktu, że największa możliwa wartość zewnętrznej wagi Viterbiego wynosi 0, a nie istnieje żadne inne górne ograniczenie na tę wartość. Oszacowanie $a = 0$ odpowiada użyciu samych tylko oszacowań b jako priorytetów wyjmowania krawędzi z agendy (algorytm równomiernego kosztu). Jako stale równe zero, oszacowanie to jest najłatwiejsze do obliczenia. Z drugiej strony, oszacowanie NULL nie daje żadnej informacji o kontekście, więc

liczba przetworzonych przez algorytm tych krawędzi z agendy, które nie należą do szukanego drzewa Viterbiego, będzie stosunkowo duża.

Na drugim krańcu wachlarza możliwych oszacowań kontekstu jest oszacowanie oparte na całkowitej wiedzy o kontekście — oszacowanie TRUE. To oszacowanie daje wartość $a(e) = \alpha(e)$ dla każdej krawędzi e i dzięki temu pozwala wyjmować z agendy tylko właściwe krawędzie, ale w praktyce jest bezużyteczne, ponieważ jest równoważne wykonywaniu całego algorytmu dla każdej krawędzi z osobna — dlatego oszczędności czasowe są wtedy żadne.

Istnieje jednak dużo oszacowań pośrednich, opartych na częściowej wiedzy. Niech $G = (V, T, R, S, P)$ będzie probabilistyczną gramatyką bezkontekstową, zaś $w = w_1 \dots w_n \in T^+$ parsowanym łańcuchem symboli terminalnych. Oznaczmy przez $X[i, j]$ aktualnie przetwarzaną krawędź, niech Y oznacza etykietę krawędzi sąsiadującej z nią bezpośrednio z jej lewej strony, zaś Z — z prawej strony. Lista przykładowych oszacowań kontekstowych przedstawia się następująco, w kolejności od najmniejszej do największej wiedzy o kontekście:

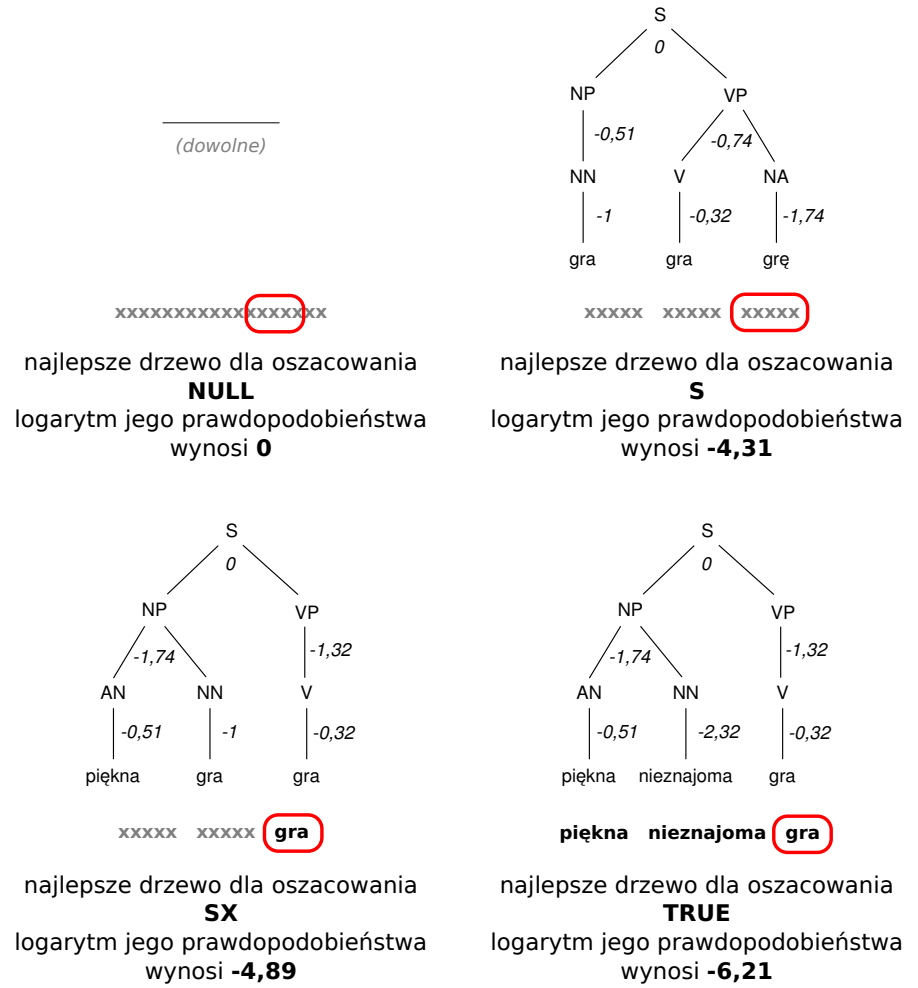
— NULL: nie mamy żadnej wiedzy na temat kontekstu,
— S_1: określona jest łączna liczba symboli terminalnych po lewej i prawej stronie bieżącej krawędzi, tj. znamy $i + n - j$,
— S: określone są liczby symboli terminalnych osobno po lewej i po prawej stronie bieżącej krawędzi, tj. znamy i oraz $n - j$,
— SX: określone są liczby symboli terminalnych osobno po lewej i po prawej stronie bieżącej krawędzi oraz jej etykieta, tj. znamy i, $n - j$ oraz X,
— SXL: określone są liczby symboli terminalnych osobno po lewej i po prawej stronie bieżącej krawędzi oraz jej etykieta, a także etykieta krawędzi bezpośrednio sąsiadującej z bieżącą po jej lewej stronie, tj. znamy i, $n - j$, X oraz Y,
— SXR: określone są liczby symboli terminalnych osobno po lewej i po prawej stronie bieżącej krawędzi oraz jej etykieta, a także etykieta krawędzi bezpośrednio sąsiadującej z bieżącą po jej prawej stronie, tj. znamy i, $n - j$, X oraz Z,
— S_1XLR: określone są łączna liczba symboli terminalnych po lewej i po prawej stronie bieżącej krawędzi oraz jej etykieta, a także etykiety obu krawędzi bezpośrednio sąsiadujących z bieżącą, tj. znamy i, $n - j$, X, Y oraz Z,
— SXLR: określone są liczby symboli terminalnych osobno po lewej i po prawej stronie bieżącej krawędzi oraz jej etykieta, a także etykiety obu krawędzi bezpośrednio sąsiadujących z bieżącą, tj. znamy i, $n - j$, X, Y oraz Z,
— TRUE: pełna wiedza o kontekście.

Wybrane przykłady oszacowań kontekstowych przedstawione są na rysunku 4.7.

4.2.3. Heurystyki oparte na rzutowaniu gramatyki

Innym sposobem na uzyskanie oszacowań dla heurystyki a jest zastosowanie rzutowania gramatyki. W oszacowaniach opartych na kontekście używaliśmy oryginalnej gramatyki, ale kontekst był niepełny. W oszacowaniach opartych na rzutowaniu gramatyki będziemy używać dokładnego kontekstu, ale gramatyka będzie uproszczona.

Definicja 4.2 (rzut gramatyki)**.** Niech $G = (V, T, R, S, P)$ będzie probabilistyczną gramatyką bezkontekstową. Niech dana będzie funkcja rzutująca $\pi\colon V \cup$



Rysunek 4.7. Przykłady (zewnętrznych) oszacowań kontekstowych. Wyróżniono symbol (krawędź), dla którego obliczane jest oszacowanie. Pokazano drzewa wyprowadzeń, które dają najwyższą zewnętrzną wagę Viterbiego przy danej wiedzy na temat kontekstu — wartość ta jest wartością oszacowania a.

$T \to N$, gdzie N jest dowolnym zbiorem symboli. Wówczas *rzutem gramatyki* G za pomocą funkcji π nazywamy gramatykę z wagami $G' = (V', T', R', S', P')$ taką, że

- $V' := \{\pi(A) \colon A \in V\} \subseteq N$,
- $T' := \{\pi(a) \colon a \in T\} \subseteq N$ jest zbiorem rozłącznym z V',
- $R' := \{\hat{\pi}(r) \colon r \in R\}$, gdzie $\hat{\pi} \colon R \to R'$ jest funkcją zdefiniowaną następująco:

$$\hat{\pi}(A \to X_1 \dots X_k) := \pi(A) \to \pi(X_1) \dots \pi(X_k)$$

  dla $A \in V$, $X_1, \dots, X_k \in V \cup T$,
- $S' := \pi(S)$,
- funkcja wagi P' jest określona następująco:

$$P'(r') := \max_{r \colon \hat{\pi}(r) = r'} P(r).$$

Rzut G' probabilistycznej gramatyki bezkontekstowej G na ogół nie musi być probabilistyczną gramatyką bezkontekstową, ponieważ funkcja wagi P' nie musi spełniać warunku sumowalności do 1 reguł o tym samym poprzedniku.

Dla każdej reguły $r \in R$ gramatyki $G = (V, T, R, S, P)$ zachodzi własność:

$$P'(r') \geqslant P(r),$$

co wynika ze sposobu określenia funkcji wagi w rzucie gramatyki.

Oznacza to, że prawdopodobieństwa drzew wyprowadzeń i łańcuchów również nie zmniejszają się w procesie rzutowania.

Z tego samego powodu zachodzi związek

$$\alpha_G(e) \leqslant \alpha_{G'}(e')$$

dla każdej krawędzi e (jeżeli e' jest jej rzutem w gramatyce G'). Dzięki temu możemy wykorzystać oszacowanie $a(e) := \alpha_{G'}(e')$ jako dopuszczalną heurystykę.

Podobnie jak w przypadku oszacowań opartych na kontekście, możemy wyróżnić cały szereg możliwych oszacowań opartych na rzutowaniu, począwszy od oszacowania trywialnego NULL opartego na rzucie stałym (wszystkie symbole gramatyki rzutujemy na jeden symbol) aż do oszacowania TRUE opartego na rzucie identycznościowym. Możliwe są też oszacowania mieszane, powstałe z połączenia oszacowań kontekstowych i opartych na rzutowaniu.

4.2.4. Przykład parsingu za pomocą algorytmu A*

Przykład 4.2 (parsing wykorzystujący algorytm A* z heurystyką SX)**.** Niech dana będzie gramatyka bezkontekstowa z przykładu 2.26 na stronie 23 oraz łańcuch *nieznajoma gra pika* .

Ponieważ stosując podejście wstępujące możemy w prosty sposób obliczać dokładną wartość wagi β, zamiast oszacowania b będziemy stosować dokładną wartość β.

Najpierw przygotowujemy pustą tablicę i agendę. Do agendy wkładamy krawędzie *nieznajoma* $[0, 1]$, *gra* $[1, 2]$ oraz *pika* $[2, 3]$. Dla każdej krawędzi obliczamy wartości β, a i $\beta+a$. Wartości wagi β dla każdej z krawędzi *nieznajoma* $[0, 1]$, *gra* $[1, 2]$ oraz *pika* $[2, 3]$ są równe 0, ponieważ krawędzie te odpowiadają symbolom terminalnym, a prawdopodobieństwo wyprowadzenia symbolu terminalnego z tego samego symbolu terminalnego wynosi 1.

Wartości $a(e)$ obliczamy jako logarytm największego z prawdopodobieństw drzew wyprowadzenia, które zawierają krawędź e. Jeden ze sposobów, jak obliczyć wartość $a(e)$ bez konieczności obliczania prawdopodobieństw wszystkich drzew zawierających krawędź e, zostanie przedstawiony w sekcji 5.2.1 (na przykładzie oszacowania SX).

Aktualna zawartość agendy uwidoczniona jest w poniższej tabeli (wytłuszczono nowo dodane krawędzie):

agenda			
krawędź e	$\beta(e)$	$a(e)$	$(\beta + a)(e)$
***nieznajoma* [0,1]**	$\log(1) = 0$	$\log(0.0202) = -5.63$	-5.63
***gra* [1,2]**	$\log(1) = 0$	$\log(0.0504) = -4.31$	$\mathbf{-4.31}$
***pika* [2,3]**	$\log(1) = 0$	$\log(0.0336) = -4.89$	-4.89

Jako pierwszą z agendy wyjmujemy krawędź *gra* $[1, 2]$, ponieważ ma największą wartość oszacowania $\beta + a$. Dodajemy tę krawędź do tablicy. Teraz zawartość tablicy jest następująca (wytłuszczono nowo dodaną krawędź):

tablica	
krawędź e	$\beta(e)$
***gra* [1,2]**	$\log(1) = 0$

Wypiszmy teraz wszystkie reguły gramatyki, które po prawej stronie mają symbol *gra*:

— $NN \rightarrow gra$,

— $V \rightarrow gra$.

Każda z powyższych reguł może być zastosowana do przetwarzanej aktualnie krawędzi, ponieważ są to reguły unarne (mają po prawej stronie tylko jeden symbol).

Konstruujemy zatem nowe krawędzie $NN\,[1,2]$ i $V\,[1,2]$. Dodajemy je do agendy i obliczamy wartości oszacowań:

$$\beta(\,NN\,[1,2]) = \log P(\,NN \rightarrow gra\,) = \log(0.5) = -1,$$
$$\beta(\,V\,[1,2]) = \log P(\,V \rightarrow gra\,) = \log(0.8) = -0.32.$$

Wartość $a(\,NN\,[1,2])$ równa jest logarytmowi prawdopodobieństwa najbardziej prawdopodobnego drzewa wyprowadzenia łańcucha $c\,NN\,d$ z symbolu początkowego S gramatyki, gdzie c i d są dowolnymi symbolami końcowymi. Podobnie oblicza się wartość $a(\,V\,[1,2])$.

Poniższa tabela przedstawia zawartość agendy po wykonaniu tego kroku:

agenda			
krawędź e	$\beta(e)$	$a(e)$	$(\beta + a)(e)$
nieznajoma $[0,1]$	$\log(1) = 0$	$\log(0.0202) = -5.63$	-5.63
***NN* [1,2]**	$\log(0.5) = -1.00$	$\log(0.0672) = -3.89$	-4.89
***V* [1,2]**	$\log(0.8) = -0.32$	$\log(0.063) = -3.99$	$\mathbf{-4.31}$
pika $[2,3]$	$\log(1) = 0$	$\log(0.0336) = -4.89$	-4.89

Znów wyjmujemy z agendy tę krawędź, dla której wartość $\beta + a$ jest największa, czyli tym razem jest to $V[1,2]$. Krawędź $V[1,2]$ wstawiamy do tablicy:

tablica	
krawędź e	$\beta(e)$
gra $[1,2]$	$\log(1) = 0$
***V* [1,2]**	$\log(0.8) = -0.32$

Ponieważ żadna z produkcji gramatyki G nie ma po prawej stronie pojedynczego symbolu V, nie zostają utworzone żadne nowe krawędzie, które można by było dorzucić do agendy.

Teraz w agendzie znajdują się dwie krawędzie, których waga jest maksymalna:

agenda			
krawędź e	$\beta(e)$	$a(e)$	$(\beta + a)(e)$
nieznajoma $[0,1]$	$\log(1) = 0$	$\log(0.0202) = -5.63$	-5.63
NN $[1,2]$	$\log(0.5) = -1.00$	$\log(0.0672) = -3.89$	$\mathbf{-4.89}$
pika $[2,3]$	$\log(1) = 0$	$\log(0.0336) = -4.89$	$\mathbf{-4.89}$

Wybieramy dowolną z nich, na przykład $NN[1,2]$:

tablica	
krawędź e	$\beta(e)$
gra $[1,2]$	$\log(1) = 0$
$V\,[1,2]$	$\log(0.8) = -0.32$
$\boldsymbol{N}$ **[1,2]**	$\log(0.5) = -1.00$

I w tym przypadku nie są tworzone żadne nowe krawędzie:

agenda			
krawędź e	$\beta(e)$	$a(e)$	$(\beta + a)(e)$
nieznajoma $[0,1]$	$\log(1) = 0$	$\log(0.0202) = -5.63$	-5.63
pika $[2,3]$	$\log(1) = 0$	$\log(0.0336) = -4.89$	$\mathbf{-4.89}$

Postępujemy dalej w sposób podobny jak do tej pory, wybierając z agendy krawędź o największej wadze i znajdując, z jakimi krawędziami z tablicy może zostać połączona w nowe krawędzie. Proces ten ilustrują tabele przedstawiające zawartości agendy i tablicy w kolejnych krokach. Wytłuszczono krawędzie nowo dodane do struktury. W agendzie wytłuszczono również wagę krawędzi o największej wartości oszacowania $\beta + a$.

tablica	
krawędź e	$\beta(e)$
gra $[1,2]$	$\log(1) = 0$
$V\,[1,2]$	$\log(0.8) = -0.32$
$N\,[1,2]$	$\log(0.5) = -1.00$
pika **[2,3]**	$\log(1) = 0$

Reguły z symbolem *pika* po prawej stronie:

— $NN \rightarrow pika$,
— $NA \rightarrow pika$,
— $V \rightarrow pika$.

Wartości β dla nowych krawędzi:

$$\beta(\,NN\,[2,3]) = \log P(\,NN \rightarrow pika\,) = \log(0.3) = -1.74,$$
$$\beta(\,NA\,[2,3]) = \log P(\,NA \rightarrow pika\,) = \log(0.2) = -2.32,$$
$$\beta(\,V\,[2,3]) = \log P(\,V \rightarrow pika\,) = \log(0.2) = -2.32.$$

Wartość oszacowania $a(\,NN\,)$ wynosi $-\infty$, ponieważ nie istnieje żadne drzewo wyprowadzenia łańcucha $a\,b\,NN$ z symbolu początkowego S (gdzie $a, b \in T$).

agenda			
krawędź e	$\beta(e)$	$a(e)$	$(\beta + a)(e)$
nieznajoma $[0,1]$	$\log(1) = 0$	$\log(0.0202) = -5.63$	-5.63
NN **[2,3]**	$\log(0.3) = -1.74$	$\log(0) = -\infty$	$-\infty$
NA **[2,3]**	$\log(0.2) = -2.32$	$\log(0.168) = -2.57$	$\mathbf{-4.89}$
V **[2,3]**	$\log(0.2) = -2.32$	$\log(0.042) = -4.57$	-6.89

tablica	
krawędź e	$\beta(e)$
gra $[1,2]$	$\log(1) = 0$
$V\,[1,2]$	$\log(0.8) = -0.32$
$N\,[1,2]$	$\log(0.5) = -1.00$
pika $[2,3]$	$\log(1) = 0$
NA **[2,3]**	$\log(0.2) = -2.32$

Istnieje tylko jedna reguła z symbolami $X\ NA$ po prawej stronie ($X \in V \cup T$). Jest to reguła $VP \rightarrow V\ NA$. Tworzymy zatem nową krawędź $VP\,[1,3]$. Obliczamy wartość β dla nowo utworzonej krawędzi:

$$\begin{aligned}\beta(\,VP\,[1,3]) &= \log(P(\,VP \rightarrow V\ NA\,) \cdot P(\,V \rightarrow gra\,) \cdot P(\,NA \rightarrow pika\,)) = \\ &= \log P(\,VP \rightarrow V\ NA\,) + \log P(\,V \rightarrow gra\,) + \log P(\,NA \rightarrow pika\,) = \\ &= \log 0.6 + \beta(\,V\,[1,2]) + \beta(\,NA\,[2,3]) = \\ &= -0.74 - 0.32 - 2.32 = -3.38.\end{aligned}$$

agenda			
krawędź e	$\beta(e)$	$a(e)$	$(\beta + a)(e)$
nieznajoma $[0,1]$	$\log(1) = 0$	$\log(0.0202) = -5.63$	-5.63
***VP* [1,3]**	$\log(0.096) = -3.38$	$\log(0.35) = -1.51$	$\mathbf{-4.89}$
$NN\,[2,3]$	$\log(0.3) = -1.74$	$\log(0) = -\infty$	$-\infty$
$V\,[2,3]$	$\log(0.2) = -2.32$	$\log(0.042) = -4.57$	-6.89

tablica	
krawędź e	$\beta(e)$
gra $[1,2]$	$\log(1) = 0$
$V\,[1,2]$	$\log(0.8) = -0.32$
$N\,[1,2]$	$\log(0.5) = -1.00$
***VP* [1,3]**	$\log(0.096) = -3.38$
pika $[2,3]$	$\log(1) = 0$
$NA\,[2,3]$	$\log(0.2) = -2.32$

Nie ma reguł, które miałyby wyłącznie symbol VP po prawej stronie. Nie tworzymy nowych krawędzi.

agenda			
krawędź e	$\beta(e)$	$a(e)$	$(\beta + a)(e)$
nieznajoma $[0,1]$	$\log(1) = 0$	$\log(0.0202) = -5.63$	$\mathbf{-5.63}$
$NN\,[2,3]$	$\log(0.3) = -1.74$	$\log(0) = -\infty$	$-\infty$
$V\,[2,3]$	$\log(0.2) = -2.32$	$\log(0.042) = -4.57$	-6.89

tablica	
krawędź e	$\beta(e)$
***nieznajoma* [0,1]**	$\log(1) = 0$
gra $[1,2]$	$\log(1) = 0$
$V\,[1,2]$	$\log(0.8) = -0.32$
$N\,[1,2]$	$\log(0.5) = -1.00$
$VP\,[1,3]$	$\log(0.096) = -3.38$
pika $[2,3]$	$\log(1) = 0$
$NA\,[2,3]$	$\log(0.2) = -2.32$

Reguły z symbolem *nieznajoma* po prawej stronie:

— $NN \rightarrow$ *nieznajoma*,

— $AN \rightarrow$ *nieznajoma*.

Wartości β dla nowych krawędzi:

$$\beta(\,NN\,[0,1]) = \log P(\,NN \rightarrow nieznajoma\,) = \log(0.2) = -2.32,$$
$$\beta(\,AN\,[0,1]) = \log P(\,AN \rightarrow nieznajoma\,) = \log(0.3) = -1.74.$$

agenda			
krawędź e	$\beta(e)$	$a(e)$	$(\beta + a)(e)$
***NN* [0,1]**	$\log(0.2) = -2.32$	$\log(0.1008) = -3.31$	$\mathbf{-5.63}$
***AN* [0,1]**	$\log(0.3) = -1.74$	$\log(0.048) = -4.38$	-6.12
NN $[2,3]$	$\log(0.3) = -1.74$	$\log(0) = -\infty$	$-\infty$
V $[2,3]$	$\log(0.2) = -2.32$	$\log(0.042) = -4.57$	-6.89

tablica	
krawędź e	$\beta(e)$
nieznajoma $[0,1]$	$\log(1) = 0$
***NN* [0,1]**	$\log(0.2) = -2.32$
gra $[1,2]$	$\log(1) = 0$
V $[1,2]$	$\log(0.8) = -0.32$
N $[1,2]$	$\log(0.5) = -1.00$
VP $[1,3]$	$\log(0.096) = -3.38$
pika $[2,3]$	$\log(1) = 0$
NA $[2,3]$	$\log(0.2) = -2.32$

Jedyną regułą, którą można zastosować do krawędzi *NN* $[0,1]$ (i ewentualnie sąsiednich) jest $NP \rightarrow NN$.

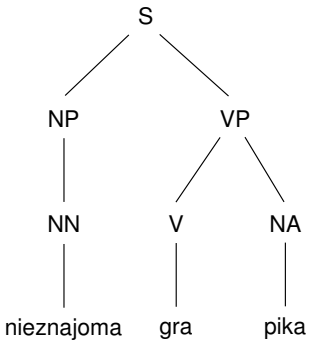
$$\beta(\, NP\,[0,1]) = \log P(\, NP \rightarrow NN\,) + \beta(\, NN\,[0,1]) = -0.51 - 2.32 = -2.83.$$

agenda			
krawędź e	$\beta(e)$	$a(e)$	$(\beta + a)(e)$
AN $[0,1]$	$\log(0.3) = -1.74$	$\log(0.048) = -4.38$	-6.12
***NP* [0,1]**	$\log(0.14) = -2.83$	$\log(0.144) = -2.80$	$\mathbf{-5.63}$
NN $[2,3]$	$\log(0.3) = -1.74$	$\log(0) = -\infty$	$-\infty$
V $[2,3]$	$\log(0.2) = -2.32$	$\log(0.042) = -4.57$	-6.89

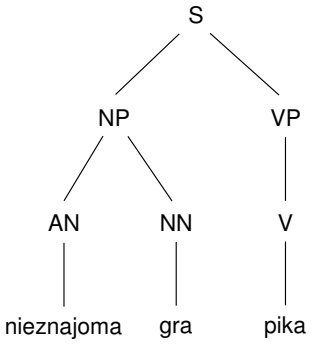
tablica	
krawędź e	$\beta(e)$
nieznajoma $[0,1]$	$\log(1) = 0$
NN $[0,1]$	$\log(0.2) = -2.32$
***NP* [0,1]**	$\log(0.14) = -2.83$
gra $[1,2]$	$\log(1) = 0$
V $[1,2]$	$\log(0.8) = -0.32$
N $[1,2]$	$\log(0.5) = -1.00$
VP $[1,3]$	$\log(0.096) = -3.38$
pika $[2,3]$	$\log(1) = 0$
NA $[2,3]$	$\log(0.2) = -2.32$

Istnieje tylko jedna reguła, która ma symbol *NP* po prawej stronie. Tą regułą jest $S \rightarrow NP\ VP$. Można ją teraz zastosować, ponieważ krawędź *VP* $[1,k]$ ($k \in \{2,3\}$) znajduje się w tablicy. Zastosowanie tej reguły powoduje utworzenie krawędzi *S* $[0,3]$ zawierającej symbol początkowy gramatyki, więc można zakończyć działanie algorytmu. Znaleziona wartość

$$\beta(\, S\,[0,3]) = \log P(\, S \rightarrow NP\ \ VP\,) + \beta(\, NP\,[0,1]) + \beta(\, VP\,[1,2]) = \\ = 0 - 2.83 - 3.38 = -6.21$$



Rysunek 4.8. Drzewo Viterbiego łańcucha *nieznajoma gra pika* uzyskane za pomocą algorytmu A*.



Rysunek 4.9. Drzewo wyprowadzenia łańcucha *nieznajoma gra pika* , które nie jest drzewem Viterbiego.

jest logarytmem prawdopodobieństwa Viterbiego łańcucha *nieznajoma gra pika* (por. wynik uzyskany algorytmem CYK w przykładzie 3.1):

$$\log 0.01344 = -6.21.$$

Ostateczna zawartość tablicy po zakończeniu działania algorytmu A*:

tablica	
krawędź e	$\beta(e)$
nieznajoma $[0,1]$	$\log(1) = 0$
NN $[0,1]$	$\log(0.2) = -2.32$
NP $[0,1]$	$\log(0.14) = -2.83$
$\boldsymbol{S}$ **[0,3]**	$\log(0.01344) = -6.21$
gra $[1,2]$	$\log(1) = 0$
V $[1,2]$	$\log(0.8) = -0.32$
N $[1,2]$	$\log(0.5) = -1.00$
VP $[1,3]$	$\log(0.096) = -3.38$
pika $[2,3]$	$\log(1) = 0$
NA $[2,3]$	$\log(0.2) = -2.32$

Przeglądając krawędzie zawarte w tablicy i reguły, które zostały użyte do ich utworzenia, możemy odtworzyć drzewo Viterbiego łańcucha *nieznajoma gra pika* (rys. 4.8). Nie różni się ono niczym od drzewa uzyskanego za pomocą algorytmu CYK.

Widać, że część krawędzi tworzących alternatywne drzewo wyprowadzenia rozważanego łańcucha (4.9) nie była przetwarzana przez algorytm A*, czego nie można powiedzieć o algorytmie CYK.

Rozdział 5

Implementacja algorytmu A* w systemie Translatica, uzasadnienie dokonanych wyborów

5.1. Rozwiązania wykorzystywane w systemie Translatica

5.1.1. System tłumaczenia automatycznego Translatica

Systemy tłumaczenia automatycznego można z grubsza podzielić na *regułowe*, czyli takie, które wykorzystują analizę składniową tłumaczonych zdań, i *statystyczne*, które opierają się na analizie statystycznej wielkich zestawów (korpusów) dwujęzycznych tekstów. Translatica jest systemem tłumaczenia regułowego, który tłumaczy teksty z języka polskiego na języki: angielski, niemiecki i rosyjski, i w drugą stronę.

W systemie Translatica wykorzystywane są dwa różne parsery. Jeden z nich służy do analizy składniowej języków: polskiego, angielskiego i rosyjskiego. Parser ten nie korzysta z gramatyk probabilistycznych, więc nie można zastosować algorytmu A* do poprawienia jego wydajności.

Do parsowania języka niemieckiego używany jest inny parser, opracowany później. Parser języka niemieckiego korzysta z wag probabilistycznych reguł, a wagi te zostały uzyskane z korpusów języka niemieckiego metodami analizy statystycznej. Z tych powodów parser Translatiki dla języka niemieckiego nadawał się do tego, aby spróbować zastosować algorytm A* do poprawienia szybkości i jakości jego działania.

5.1.2. Algorytm parsingu systemu Translatica

Parser języka niemieckiego stosowany w systemie Tranaslatica (którego będę w tym rozdziale nazywał krótko *parserem Translatiki*) jest parserem tablicowym będącym implementacją algorytmu CYK. Dla uproszczenia implementacji przyjmuje się, że używana gramatyka jest w postaci zbinaryzowanej (w sensie definicji 2.42 ze strony 15). Nie jest to szczególnym ograniczeniem, ponieważ zgodnie z twierdzeniem 2.4 dla dowolnej gramatyki bezkontekstowej możemy znaleźć równoważną jej zbinaryzowaną gramatykę bezkontekstową. Szkic algorytmu binaryzacji gramatyki bezkontekstowej został nakreślony w dowodzie tegoż twierdzenia.

Wagi probabilistyczne wykorzystywane są podczas parsowania, aby nadać niektórym regułom wyższe priorytety niż innym.

Parser Translatiki korzysta z typowych struktur danych właściwych parsingowi tablicowemu, takich jak tablica i agenda. Informacje o symbolach gramatyki, ich zakresach w budowanym drzewie wyprowadzenia parsowanego łańcucha i wagach zastosowanych reguł reprezentuje struktura danych nazywana *krawędzią*. Jest to czwórka uporządkowana (X, i, j, W), w której:

— X jest symbolem gramatyki,
— (i, j) jest zakresem symbolu X w drzewie wyprowadzenia,
— W jest wagą przypisaną wierzchołkowi drzewa wyprowadzenia o etykiecie $X[i, j]$ w trakcie działania algorytmu.

Poniżej przedstawiony jest uproszczony algorytm parsingu stosowany w systemie Translatica.

Algorytm 5.1 (algorytm parsingu tablicowego stosowany w systemie Translatica)**.** Niech dana będzie zbinaryzowana gramatyka bezkontekstowa z wagami $G = (V, T, R, S, P)$ oraz łańcuch $w = w_1 \dots w_n \in T^+$.

1. Przygotuj tablicę i włóż do niej krawędzie $(w_1, 0, 1, 0)$, $(w_2, 1, 2, 0)$, $\dots$, $(w_n, n-1, n, 0)$.
2. Przygotuj agendę i włóż do niej krawędzie $(w_1, 0, 1, 0)$, $(w_2, 1, 2, 0)$, $\dots$, $(w_n, n-1, n, 0)$.
3. Powtarzaj dopóki agenda nie będzie pusta:
   a) wyjmij z agendy kolejną krawędź (Z, i, j, W);
   b) dla każdej reguły unarnej $A \to Z \in R$, jeżeli krawędzi odpowiadającej wierzchołkowi $A[i, j]$ drzewa rozkładu jeszcze nie ma w tablicy:
      i. dodaj do tablicy krawędź $(A, i, j, W + P(A \to Z))$,
      ii. dodaj do agendy krawędź $(A, i, j, W + P(A \to Z))$;
   c) dla każdej reguły binarnej $A \to XZ \in R$ takiej, że krawędź (X, k, i, W') jest w tablicy, ale krawędzi odpowiadającej wierzchołkowi $A[k, j]$ jeszcze nie ma w tablicy:
      i. dodaj do tablicy krawędź $(A, k, j, W + W' + P(A \to XZ))$,
      ii. dodaj do agendy krawędź $(A, k, j, W + W' + P(A \to XZ))$;
   d) dla każdej reguły binarnej $A \to ZY \in R$ takiej, że krawędź (Y, j, l, W'') jest w tablicy, ale krawędzi odpowiadającej wierzchołkowi $A[i, l]$ jeszcze nie ma w tablicy:
      i. dodaj do tablicy krawędź $(A, i, l, W + W'' + P(A \to ZY))$,
      ii. dodaj do agendy krawędź $(A, i, l, W + W'' + P(A \to ZY))$;
4. Po zakończeniu działania algorytmu tablica zawiera strukturę drzewa wyprowadzenia wejściowego łańcucha. Jeżeli jest to możliwe, tj. jeśli $w \in L(G)$, to korzeniem otrzymanego drzewa jest symbol początkowy S gramatyki.

W powyższym algorytmie kroki 3a–3d składają się na operację, którą będziemy nazywać *obrotem* parsera.

5.1.3. Gramatyki atrybutowe

W parserze systemu Translatica wykorzystywane są nie tyle gramatyki bezkontekstowe, co gramatyki atrybutowe — struktury będące rozszerzeniem gramatyk bezkontekstowych. Gramatyki atrybutowe lepiej nadają się do analizy języków naturalnych niż zwyczajne gramatyki bezkontekstowe, ponieważ umożliwiają odzwierciedlenie związków pomiędzy poszczególnymi częściami zdania, wyrażających się przez uzgadnianie form wyrazów.

W tym rozdziale przedstawię jedynie zarys pojęcia gramatyki atrybutowej w zakresie koniecznym do zrozumienia zasad działania parsera Translatiki. Szczegółową analizę gramatyk atrybutowych i ich własności można znaleźć m.in. w [13].

Definicja 5.1 (wyrażenie atrybutowe)**.** Niech $r = X_0 \to X_1 \dots X_k \in R$ będzie regułą produkcji gramatyki bezkontekstowej $G = (V, T, R, S)$. Każdy napis, który ma jedną z poniżej podanych postaci:

(1) $[i].\mathbf{a} = \mathbf{x}$, gdzie $i \in \{0, 1, \dots, k\}$, $X_i \in V$, $\mathbf{a} \in \mathcal{A}$, $\mathbf{x} \in \mathcal{V}_\mathbf{a}$,
(2) $[i].\mathbf{a} = [j].\mathbf{a}$, gdzie $i, j \in \{0, 1, \dots, k\}$, $X_i, X_j \in V$, $\mathbf{a} \in \mathcal{A}$,

nazywamy *wyrażeniem atrybutowym dla reguły* r.

Definicja 5.2 (gramatyka atrybutowa)**.** *Gramatyką atrybutową* nazywamy strukturę $G = (V, T, R, S, \mathcal{A}, \mathcal{V}, \mathcal{E})$, gdzie:

— (V, T, R, S) jest gramatyką bezkontekstową,
— $\mathcal{A}$ jest skończonym *zbiorem atrybutów*,
— $\mathcal{V} := \{\mathcal{V}_\mathbf{a}\}_{\mathbf{a}\in\mathcal{A}}$ jest rodziną skończonych zbiorów wartości poszczególnych atrybutów,
— $\mathcal{E} := \{\mathcal{E}_r\}_{r\in R}$ jest rodziną skończonych *zbiorów wyrażeń atrybutowych*, tj. każdej regule produkcji $r \in R$ przyporządkowany jest jeden zbiór, którego elementy są wyrażeniami atrybutowymi dla tej reguły.

Wyrażenia atrybutowe ze zbioru $\mathcal{E}_r$ nazywamy *wyrażeniami atrybutowymi powiązanymi z regułą* r. Zwróćmy uwagę, że nie każde wyrażenie atrybutowe dla reguły r jest powiązane z regułą r.

Wyprowadzeniami w gramatykach atrybutowych rządzą nieco inne prawa niż w zwyczajnych gramatykach bezkontekstowych. Każdemu symbolowi pomocniczemu użytemu w wyprowadzeniu przypisane są konkretne wartości atrybutów ze zbioru $\mathcal{A}$. Produkcja może zostać użyta w wyprowadzeniu jedynie wtedy, gdy wartości atrybutów symboli w niej występujących spełniają wszystkie wyrażenia atrybutowe powiązane z tą produkcją.

Jeżeli wyrażenie atrybutowe jest postaci 1 z definicji 5.1, to oznacza, że aby produkcja mogła zostać użyta, aktualna wartość atrybutu $\mathbf{a}$ dla symbolu X_i musi wynosić $\mathbf{x}$. Z kolei wyrażenie atrybutowe postaci 2 oznacza, że aktualna wartość atrybutu $\mathbf{a}$ dla symbolu X_i musi być równa aktualnej wartości atrybutu $\mathbf{a}$ dla symbolu X_j.

Można zdefiniować również gramatyki atrybutowe z wagami, a nawet probabilistyczne gramatyki atrybutowe:

Definicja 5.3 (gramatyka atrybutowa z wagami, probabilistyczna gramatyka atrybutowa)**.** Strukturę $G = (V, T, R, S, P, \mathcal{A}, \mathcal{V}, \mathcal{E})$ jest gramatyką atrybutową z wagami (probabilistyczną gramatyką atrybutową), jeżeli:

— $(V, T, R, S, \mathcal{A}, \mathcal{V}, \mathcal{E})$ jest gramatyką atrybutową,
— (V, T, R, S, P) jest gramatyką bezkontekstową z wagami (probabilistyczną gramatyką bezkontekstową).

Gramatyki atrybutowe z wagami są wykorzystywane w implementacji parsera systemu Translatica.

Twierdzenie 5.1. *Dla każdej gramatyki atrybutowej G można znaleźć gramatykę bezkontekstową G' taką, że $L(G') = L(G)$.*

Dowód. Ponieważ nie zdefiniowaliśmy ściśle, jak należy rozumieć $L(G)$, gdy G jest gramatyką atrybutową, przedstawiony zostanie jedynie szkic konstrukcji.

Niech $G = (V, T, R, S, \mathcal{A}, \mathcal{V}, \mathcal{E})$ będzie gramatyką atrybutową. Jeżeli $\mathcal{A} = \{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m\}$, to niech

$$V' := \{(A, \mathbf{a}_1, \mathbf{x}_1, \dots, \mathbf{a}_m, \mathbf{x}_m) \colon A \in V,\ \mathbf{x}_1 \in \mathcal{V}_{\mathbf{a}_1}, \dots,\ \mathbf{x}_m \in \mathcal{V}_{\mathbf{a}_m}\}$$

będzie zbiorem symboli terminalnych z ustalonymi atrybutami (na wszystkie możliwe sposoby).

Niech R' będzie zbiorem składającym się ze wszystkich produkcji na symbolach z ustalonymi atrybutami (ze zbiorów V' i T), które są dopuszczalne przez gramatykę G. Niech w zbiorze R' znajdują się również produkcje

$$S' \to (S, \mathbf{a}_1, \mathbf{x}_1, \ldots, \mathbf{a}_m, \mathbf{x}_m),$$

gdzie S' jest nowym symbolem początkowym, a symbole $(S, \mathbf{a}_1, \mathbf{x}_1, \ldots, \mathbf{a}_m, \mathbf{x}_m)$ odzwierciedlają wszystkie dopuszczalne wartości atrybutów starego symbolu początkowego S.

Wówczas $G' := (V', T, R', S')$ jest szukaną gramatyką bezkontekstową. □

5.1.4. Wykorzystanie maszyny stosowej w parserze

Znajdowanie, w jaki sposób krawędź wyjęta z agendy może być wbudowana w drzewo rozbioru wejściowego łańcucha, polega na sprawdzeniu, jak można daną krawędź połączyć z sąsiednimi krawędziami znajdującymi się już w budowanym drzewie. Zadanie to polega nie tylko na znalezieniu odpowiedniej reguły, za pomocą której można połączyć dwie krawędzie (dwa symbole), ale też na sprawdzeniu spełnialności wyrażeń atrybutowych powiązanych z tą regułą.

Reguły wraz z odpowiadającymi im wyrażeniami atrybutowymi są wpierw kompilowane do postaci list instrukcji dla maszyny stosowej. Te listy instrukcji będziemy nazywać *wyrażeniami regułowymi*. Aby znaleźć, w jaki sposób można połączyć dane krawędzie, maszyna stosowa jest uruchamiana na danych krawędziach dla każdego wyrażenia regułowego. Podczas przetwarzania wyrażenia regułowego, wartości atrybutów poszczególnych symboli są kładzione i zdejmowane ze stosu zgodnie z instrukcjami zawartymi w tym wyrażeniu regułowym.

5.2. Implementacja algorytmu A* w parserze systemu Translatica

5.2.1. Implementacja algorytmu A* z heurystyką SX

W artykule [8] przedstawiono wyniki badań, które pokazują, że zastosowanie algorytmu A* z odpowiednio dobraną heurystyką może przyspieszyć parsing przez ograniczenie liczby niepotrzebnie przetwarzanych krawędzi.

Do implementacji w systemie Translatica wybraliśmy heurystykę SX z uwagi na przewidywaną stosunkowo dużą skuteczność (stosunkowo niewiele przetwarzanych niepotrzebnych krawędzi) przy jednocześnie prostej implementacji i krótkim oczekiwanym czasie obliczeń wstępnych.

Za każdym razem, gdy krawędź (e) wyjmowana jest z agendy, odczytywane są wartości aktualnej wewnętrznej wagi Viterbiego krawędzi $(\beta(e))$ oraz oszacowania wagi zewnętrznej na podstawie heurystyki $(a(e))$. Suma tych dwóch wartości $(a(e) + \beta(e))$ używana jest jako priorytet kolejności wyjmowania krawędzi z agendy — wyjmowana jest krawędź e, dla której suma $a(e) + \beta(e)$ jest największa.

Wartość heurystyki dla danego kontekstu obliczana jest jedynie wtedy, gdy jest to konieczne. Za każdym razem obliczona wartość heurystki dla danego

kontekstu zapisywana jest w pamięci. Jeżeli potrzebna jest wartość heurystyki dla kontekstu, dla którego była już wcześniej obliczona, obliczenia nie są przeprowadzane ponownie, lecz odczytywana i zwracana jest wartość zapisana w pamięci. Taki sposób postępowania nazywa się *memoizacją*.

Do obliczenia zewnętrznego oszacowania SX dla danego symbolu i jego kontekstu potrzebne jest obliczenie oszacowań wewnętrznych SX dla kontekstów zależnych. W razie potrzeby odpowiednia procedura (procedura 5.2) wywoływana jest rekurencyjnie.

Algorytmy obliczania oszacowań SX: wewnętrznego i zewnętrznego, są przedstawione poniżej. Zaznaczanie symbolu jako „użytego" ma na celu zapobieżeniu zapętleniu algorytmu przy rekurencyjnym wywoływaniu procedur dla reguł unarnych.

Procedura 5.2 (obliczanie wewnętrznego oszacowania SX)**.** Sposób wywołania funkcji:

$$\mathrm{wewn}_{\mathrm{SX}}(X, l).$$

Argumentami procedury są symbol $X \in V \cup T$ i długość zakresu l.

1. Jeżeli obliczono wcześniej wartość wewnętrznego oszacowania SX dla symbolu X i długości zakresu l, to zwróć jako wynik wartość przechowaną w pamięci.
2. Jeżeli $l \leqslant 1$ oraz X jest symbolem końcowym, to zapisz w pamięci i zwróć jako wynik funkcji wartość 0.
3. $s := -\infty$.
4. Znajdź wszystkie reguły postaci $X \to Y \in R$, gdzie $Y \in V \cup T$, i dla każdej takiej reguły wykonuj:
   — jeżeli symbol Y nie został zaznaczony jako „użyty":
      a) oznacz symbol Y jako „użyty",
      b) $c := \mathrm{wewn}_{\mathrm{SX}}(Y, l) + \log P(X \to Y)$,
      c) jeśli $c > s$, to $s := c$.
5. Wyczyść listę symboli „użytych".
6. Dla i od 1 do $l - 1$ wykonuj:
   — dla każdej reguły postaci $X \to YZ \in R$, $Y, Z \in V \cup T$, wykonuj:
      a) $c := \mathrm{wewn}_{\mathrm{SX}}(Y, i) + \mathrm{wewn}_{\mathrm{SX}}(Z, l - i) + \log P(X \to YZ)$,
      b) jeśli $c > s$, to $s := c$.
7. Zapisz w pamięci wartość s i zwróć ją jako wynik funkcji.

Algorytm 5.3 (obliczanie zewnętrznego oszacowania SX)**.** Sposób wywołania funkcji:

$$\mathrm{zewn}_{\mathrm{SX}}(X, l, m).$$

Argumentami procedury są symbol $X \in V \cup T$ oraz długości zakresów: lewego l i prawego m.

1. Jeżeli obliczono wcześniej wartość zewnętrznego oszacowania SX dla symbolu X i długości zakresów: lewego l i prawego m, to zwróć jako wynik wartość przechowaną w pamięci.
2. Jeżeli $l + m \leqslant 0$ oraz X jest symbolem początkowym, to zapisz w pamięci i zwróć jako wynik funkcji wartość 0.
3. $s := -\infty$.
4. Dla każdej reguły postaci $A \to X \in R$, $A \in V$, wykonuj:

— jeżeli symbol A nie został zaznaczony jako „użyty”:
  a) oznacz symbol A jako „użyty”,
  b) $c := \text{zewn}_{\text{SX}}(X, l, m) + \log P(A \to X)$,
  c) jeśli $c > s$, to $s := c$.

5. Wyczyść listę symboli „użytych”.
6. Dla i od 1 do l wykonuj:
   — dla każdej reguły postaci $A \to YX \in R$, $A \in V$, $Y \in V \cup T$, wykonuj:
     a) $c := \text{wewn}_{\text{SX}}(Y, i) + \text{zewn}_{\text{SX}}(A, l - i, m) + \log P(A \to YX)$,
     b) jeśli $c > s$, to $s := c$.
7. Dla i od 1 do m wykonuj:
   — dla każdej reguły postaci $A \to XZ \in R$, $A \in V$, $Z \in V \cup T$, wykonuj:
     a) $c := \text{wewn}_{\text{SX}}(Z, i) + \text{zewn}_{\text{SX}}(A, l, m - i) + \log P(A \to XZ)$,
     b) jeśli $c > s$, to $s := c$.
8. Zapisz w pamięci wartość s i zwróć ją jako wynik funkcji.

5.2.2. Rzutowanie gramatyki atrybutowej

W celu dalszego przyspieszenia obliczeń, postanowiliśmy dodatkowo, oprócz heurystyki SX, zastosować rzutowanie gramatyki. Ponieważ gramatyka stosowana w parserze Translatiki jest gramatyką atrybutową posiadającą wiele symboli pomocniczych i końcowych, bez rzutowania większość kontekstów (nawet dla tak zgrubnego oszacowania jak SX) byłaby unikalna. Dlatego zysk z zastosowania memoizacji byłby niewielki.

Aby zrzutować gramatykę atrybutową z wagami, należy najpierw zamienić ją na gramatykę bezkontekstową z wagami za pomocą procedury podobnej do tej opisanej w sekcji 5.1.3. Należy zwrócić uwagę przy tym na odpowiednie przyporządkowanie wag wynikowym regułom, zwłaszcza jeśli jest to gramatyka probabilistyczna.

Najprostszą metodą rzutowania gramatyki atrybutowej wydaje się rzutowanie przez odrzucenie atrybutów. Każdy zbiór symboli utworzony z pojedynczego symbolu w powyższej procedurze zamiany gramatyki atrybutowej na bezkontekstową rzutujemy z powrotem na pojedynczy symbol. Zastosowanie tej metody niesie ze sobą ryzyko zbytniego zubożenia gramatyki po zrzutowaniu, co może być powodem zbyt zgrubnych oszacowań heurystyki.

Inną metodą jest rzutowanie gramatyki atrybutowej przez ustalenie wartości wybranych atrybutów. Wówczas rzutujemy na jeden symbol wszystkie symbole powstałe przez ustalenie wartości jednego lub kilku atrybutów pojedynczego symbolu gramatyki atrybutowej. Rzutowanie przez odrzucenie atrybutów można traktować jako wariant tej metody, w którym zbiór wybranych do rzutowania atrybutów jest zbiorem pustym.

Algorytm rzutowania przez ustalenie wartości niektórych atrybutów w istocie niewiele różni się od algorytmu zamiany gramatyki atrybutowej na bezkontekstową. Inny jest jedynie zbiór atrybutów, których używa się do wyznaczenia nowego zbioru symboli pomocniczych i nowego zbioru produkcji.

Istotnym problemem, który się napotyka, jest wyznaczenie zbioru wartości $\mathcal{V}_{\mathbf{a}}$ każdego rzutowanego atrybutu $\mathbf{a} \in \mathcal{A}$, jeżeli zbiory te nie są dane z góry. Można do tego celu wykorzystać zmodyfikowaną maszynę stosową przedstawioną w sekcji 5.1.4. Przetwarzane jest każde wyrażenie regułowe, i ilekroć napotkana

jest instrukcja, która czyta bądź zapisuje na stosie wartość atrybutu, wartość ta dopisywana jest do listy powiązanej z danym atrybutem.

5.2.3. Implementacja algorytmu A* z heurystyką NULL

Dla celów testowych i porównawczych oprócz heurystyki SX została zaimplementowana heurystyka NULL. Wartość funkcji heurystyki NULL wynosi 0 dla dowolnej krawędzi. Zaletą tej heurystyki jest zatem łatwość jej obliczania — do uszeregowania krawędzi w agendzie wystarczy jedynie znać wartość wewnętrznej wagi Viterbiego β, którą można obliczyć w sposób przedstawiony w przykładzie 4.2.

Z drugiej strony należałoby oczekiwać, że algorytm korzystający z heurystyki NULL będzie przetwarzał więcej nieużytecznych krawędzi[1] niż algorytm wykorzystujący heurystykę SX. Powinno to spowodować, że pomimo bardziej skomplikowanej obliczeniowo heurystyki, algorytm parsowania z heurystyką SX może działać szybciej niż algorytm parsowania z heurystyką NULL.

[1] Nieużytecznych krawędzi — tj. takich, które nie znajdą się w ostatecznym drzewie Viterbiego parsowanego łańcucha.

Rozdział 6

Wyniki i wnioski

6.1. Porównanie dotychczasowego parsera i parsera wykorzystującego algorytm A* pod względem poprawności i efektywności

6.1.1. Przeprowadzone testy

Parser języka niemieckiego Translatiki posiada mechanizm, który zapobiega temu, żeby parsowanie pojedynczego zdania trwało zbyt długo. Od parametru zwanego *progiem* zależy, ile *obrotów* (patrz algorytm 5.1 na stronie 61) może wykonać parser podczas parsowania jednego zdania. Im wyższy próg, tym większa dopuszczalna liczba obrotów parsera. W parserze Translatiki wartość progu ustawiona jest domyślnie na 50000.

Do testowania posłużył zestaw 100 zdań języka niemieckiego o różnej długości i o różnej tematyce. Wykonano testy tłumaczeń tych zdań z języka niemieckiego na polski w konfiguracjach, które różniły się ze względu na następujące parametry:

— zastosowany algorytm:
  — dotychczasowy parser Translatiki,
  — parser wykorzystujący algorytm A* z heurystyką SX,
  — parser wykorzystujący algorytm A* z heurystyką NULL;
— wartość progu:
  — 50000,
  — 10000;
— sposób dostarczania danych wejściowych (zdań):
  — pojedyncze zdania,
  — zestawy kilku zdań,
  — zestaw wszystkich zdań,
  — zestawy zdań powtórzonych wielokrotnie.

Zróżnicowanie sposobów dostarczania danych miało służyć zbadaniu wpływu memoizacji na szybkość parsowania. Po każdym uruchomieniu systemu tłumaczącego zmemoizowane wartości trzeba zapisywać od nowa, dlatego zdanie parsowane kolejny raz powinno zostać przetworzone szybciej niż zdanie parsowane po raz pierwszy.

6.1.2. Szybkość tłumaczenia

Tabela 6.1 przedstawia średnie czasy przetłumaczenia testowego zestawu 100 zdań niemieckich dla różnych wartości progów i dla różnych zastosowanych heurystyk.

Tabela 6.1. Średnie czasy tłumaczeń — pojedynczy zestaw 100 zdań.

próg	50000		10000	
czas	bezwzgl.	wzgl.	bezwzgl.	wzgl.
bez A*	1 min 9 s	1.00	36 s	1.00
z A*, heur. SX	2 min 2 s	1.77	52 s	1.44
z A*, heur. NULL	1 min 20 s	1.16	37 s	1.03

Tabela 6.2. Średnie czasy tłumaczeń — zestaw 100 zdań pięciokrotnie powtórzonych.

próg	50000		10000	
czas	bezwzgl.	wzgl.	bezwzgl.	wzgl.
bez A*	5 min 52 s	1.00	2 m 56 s	1.00
z A*, heur. SX	11 min 50 s	2.02	4 m 32 s	1.55
z A*, heur. NULL	6 min 55 s	1.18	3 m 6 s	1.06

Tabela 6.2 przedstawia średnie czasy przetłumaczenia zestawu testowego złożonego z pięciokrotnie powtórzonych 100 zdań użytych w poprzednim teście.

Na rysunku 6.1 na stronie 70 przedstawiono z kolei czasy tłumaczeń, kiedy każde zdanie z zestawu testowego tłumaczone było oddzielnie.

Okazuje się, że przy zastosowanej implementacji algorytmu A* nie udało się uzyskać większej szybkości tłumaczenia niż dla dotychczasowego parsera, który z algorytmu A* nie korzystał. Heurystyka NULL okazała się być szybsza od heurystyki SX i osiągnęła czasy niewiele tylko gorsze od czasów osiąganych przez dotychczasowy parser. Widać to szczególnie wyraźnie dla testów przy progu 10000, zwłaszcza tych, które nie zawierały powtórzonych zdań. Na rysunku 6.1 krzywa regresji liniowej przy progu 10000 dla heurystyki NULL pokrywa się z analogiczną krzywą dla dotychczasowego parsera.

Przewaga heurystyki NULL nad heurystyką SX może również wynikać z pewnej cechy specyficznej dla parsera Translatiki. Otóż w gramatyce używanej w parserze Translatiki nie ma wyróżnionego symbolu początkowego. Zdanie parsowane jest metodą wstępującą — krótsze krawędzie łączone są w dłuższe w sposób zachłanny, dopóki jest to możliwe. Takie postępowanie ma tę zaletę, że pozwala na parsowanie fraz, które nie są pełnymi zdaniami, lecz jedynie fragmentami zdań. Niestety, nie jest to rozwiązanie korzystne z punktu widzenia możliwości implementacyjnych dla algorytmu A*, ponieważ odróżnienie symbolu początkowego (lub symboli początkowych) jest konieczne do prawidłowego obliczenia heurystyki. Można oczywiście niektóre z wykorzystywanych symboli uznać arbitralnie za symbole początkowe (to właśnie rozwiązanie zostało zaimplementowane), jednak w ten sposób niektóre możliwe drzewa wyprowadzenia parsowanego łańcucha nie zostaną uwzględnione przy obliczaniu heurystyki. Z kolei uczynienie wszystkich symboli pomocniczych symbolami początkowymi sprawiłoby, że obliczanie heurystyki straciłoby sens.

6.1.3. Jakość tłumaczenia

Porównanie tłumaczeń dokonanych przez algorytmy: dotychczasowy, A* z heurystyką SX i A* z heurystyką NULL, przy jednakowym progu, nie wykazuje znaczących różnic w tłumaczeniu. Na ogół tłumaczenia nie różnią się, a jeśli już, to nieznacznie, na przykład pojedynczym wyrazem.

Zdarzają się zdania, których tłumaczenia przy pomocy algorytmu A* dla progu 10000 są identyczne jak tłumaczenia dokonane za pomocą dotychczasowego parsera dla progu 50000. Na przykład niemieckie zdanie *Auf dem Weg nach Bremen trifft der Esel einen Hund* ('Na drodze do Bremy osioł spotyka psa') zostało przetłumaczone przez dotychczasowy algorytm z progiem 10000 jako *Na drodze do Bremie osioł trafia psa*. Zarówno parser niekorzystający z A* z progiem ustawionym na 50000, jak i parsery korzystające z algorytmu A* przetłumaczyły to zdanie jako *Na drodze do Bremy osioł trafia psa*, co jest bliższe oczekiwanemu rezultatowi.

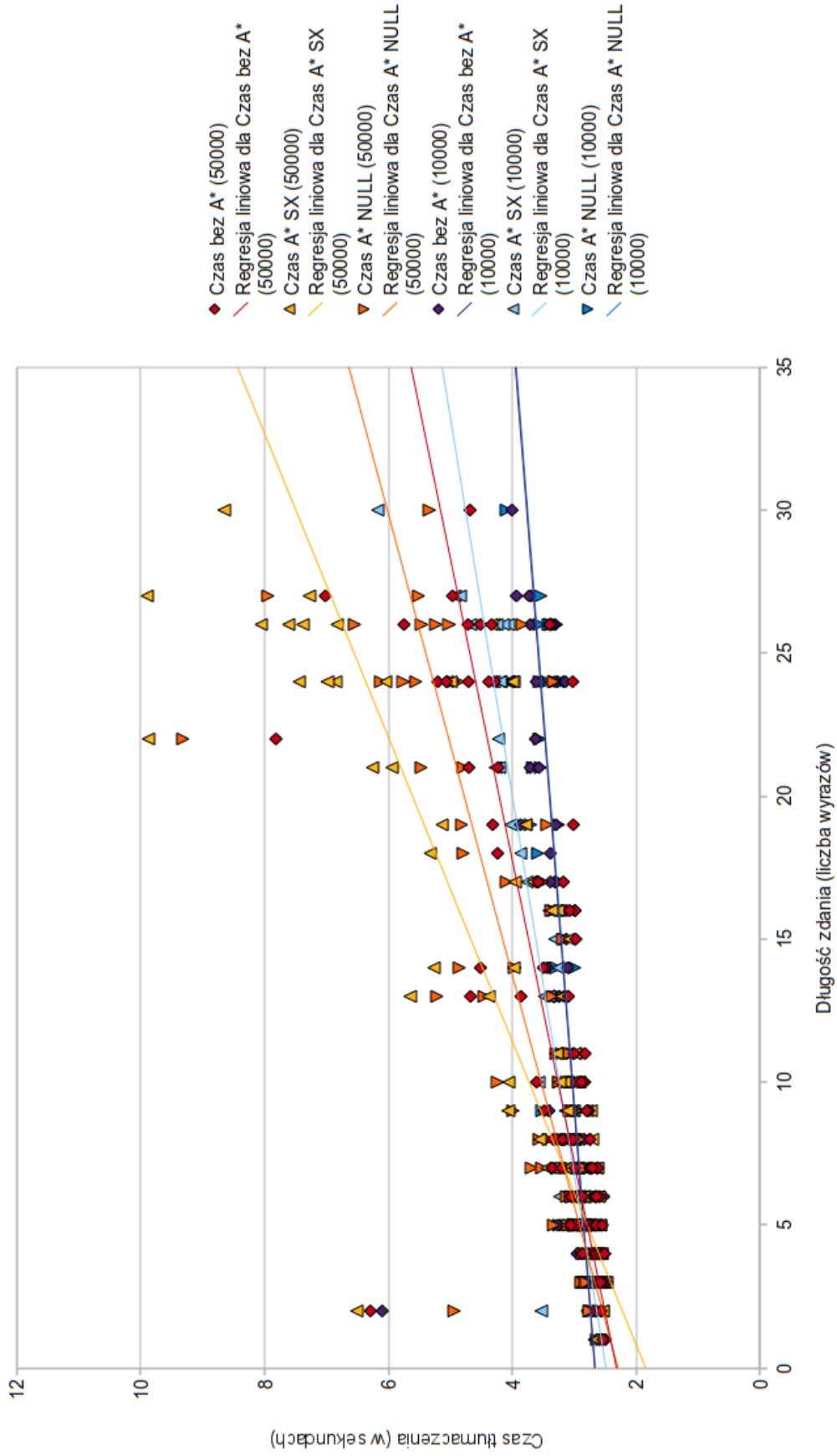
Takie przypadki mogą wskazywać na to, że rzeczywiście parser korzystający z algorytmu A* znajduje lepsze tłumaczenie w mniejszej liczbie kroków, lecz kroki te trwają dłużej przy obecnej implementacji, co jest przyczyną dłuższego czasu tłumaczenia. Być może poprawa implementacji pozwoliłaby uzyskać czasy lepsze niż dla dotychczasowego algorytmu.

6.2. Ocena przydatności zastosowanych rozwiązań i perspektywy na przyszłość

Przeprowadzona powyżej analiza wskazuje, że projekt polegający na zaimplementowaniu algorytmu A* dla parsera Translatiki nie zakończył się pełnym sukcesem. Zaimplementowanie algorytmu A* nie przyniosło oczekiwanego przyspieszenia procesu tłumaczenia ani poprawy jego jakości.

Jedną z przyczyn tego stanu rzeczy jest specyfika samego parsera, który nie korzysta z klasycznej gramatyki bezkontekstowej. Implementacja w pełni działającego algorytmu A* wymagałaby dokonania większych zmian w mechanizmie działania pasera Translatiki niż to pierwotnie zakładano.

Istnieją jednak przesłanki, które pozwalają sądzić, że po dokonaniu pewnych modyfikacji parser wykorzystujący algorytm A* będzie rzeczywiście działać szybciej i bardziej efektywnie. Być może lepsza implementacja algorytmu pozwoli skrócić czas wykonywania poszczególnych jego kroków, a w związku z tym i czas wykonywania tłumaczenia.



Rysunek 6.1. Porównanie szybkości działania algorytmów tłumaczenia.

Rozdział 7

Podsumowanie

Przedmiotem niniejszej pracy był opis algorytmów parsingu probabilistycznych gramatyk bezkontekstowych. Omówione zostały najważniejsze fakty dotyczące probabilistycznych gramatyk bezkontekstowych i przedstawione najważniejsze algorytmy ich parsingu. Szczególną uwagę poświęcono algorytmowi A*.

W ramach części projektowej pracy algorytm A* został zaimplementowany w parserze systemu Translatica. Praca zawiera opis implementacji i ważniejszych kwestii, które należało rozwiązać w związku z implementacją. W pracy przedstawione zostały wyniki porównania jakości i efektywności działania zaimplementowanego algorytmu i dotychczasowego parsera wykorzystywanego w systemie, a także wnioski płynące z tego porównania.

Bibliografia

[1] Sharon A. Caraballo, Eugene Charniak, "Figures of merit for best-first probabilistic chart parsing", [w:] *Computational Linguistics* 24, s. 275-298, 1996.

[2] Thomas H. Cormen, Charles E. Leierson, Ronald L. Rivest, Clifford Stein, *Wprowadzenie do algorytmów*, Wydawnictwa Naukowo-Techniczne, Warszawa 2005.

[3] Jay Earley, "An efficient context-free parsing algorithm", [w:] *Communications of the Association for Computing Machinery* 13 (2), s. 94-102, 1970.

[4] Heidi Jennifer Fox, *Lexicalized, Edge-Based, Best-First Chart Parsing*, MIT, 1991.

[5] Dick Grune, Ceriel Jacobs, *Parsing Techniques. A Practical Guide*, Amsterdam 1998.

[6] John E. Hopcroft, Jeffrey D. Ullman, *Wprowadzenie do teorii automatów, języków i obliczeń*, Wydawnictwo Naukowe PWN, Warszawa 2003.

[7] Daniel Jurafsky, James H. Martin, *Speech and Language Processing*, Prentice Hall, New Jersey 2000.

[8] Dan Klein, Christopher D. Manning, "A* parsing: fast exact Viterbi parse selection" [w:] *Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics*, 2003.

[9] Mirosław Krzyśko, *Wykłady z teorii prawdopodobieństwa*, UAM, Poznań 1997.

[10] Christopher D. Manning, Hinrich Schütze, *Foundations of Statistical Natural Language Processing*, The MIT Press, London 1999.

[11] Adwait Ratnaparkhi, "Learning to parse natural language with maximum entropy models", [w:] *Machine Learning* 34 (1-3), s. 151-175, 1999.

[12] Brian Roark, "Probabilistic top-down parsing and language modeling", [w:] *Computational Linguistics* 27, s. 249-276, 2001.

[13] Mads Rosendahl, "Strictness analysis for attribute grammars", [w:] *Programming Language Implementation and Logic Programming* 631, s. 145-157, 1992.

[14] Kenneth A. Ross, Charles R. B. Wright, *Matematyka dyskretna*, Wydawnictwo Naukowe PWN, Warszawa 2003.

[15] Andreas Stolcke, "An efficient probabilistic context-free parsing algorithm that computes prefix probabilities", [w:] *Computational Linguistics* 21 (2), s. 165-202, 1995.

[16] Robin J. Wilson, *Wprowadzenie do teorii grafów*, Wydawnictwo Naukowe PWN, Warszawa 1998.