Uniwersytet im. Adama Mickiewicza w Poznaniu

Wydział Matematyki i Informatyki

Praca magisterska

Zastosowanie automatów skończonych w przetwarzaniu archiwalnych tekstów języka polskiego

Autor:

Patryk Bernacki

Numer albumu: 383922

Kierunek: Informatyka

Promotor:

prof. UAM dr hab. Krzysztof Jassem

Poznań, czerwiec 2017

Streszczenie

Praca poświęcona jest problemowi przetwarzania tekstów archiwalnych języka polskiego na język współczesny. Opisany jest w niej proces tworzenia programu, który realizuje to zadanie.

W początkowych rozdziałach przedstawiona jest część teoretyczna związana z tematem pracy - rozwinięte są w niej podstawowe pojęcia, takie jak *automaty* i *transduktory*. W kolejnej części przybliżono narzędzia, które zostały wykorzystane podczas budowy programu, wraz z ich opisem i przykładami użycia.

Kolejne rozdziały omawiają temat w oparciu o badania własne. Omówienie aplikacji zawiera m.in. kluczowe fragmenty kodów, opisy algorytmów składających się na program do automatycznego przekładu tekstów oraz informację, jak należy z tego programu korzystać. Dokonano również ewaluacji rozwiązania na podstawie przykładowego tekstu z użyciem miary jakości.

Abstract

The thesis is devoted to the problem of processing archival texts of Polish into the contemporary language. It describes the process of creating a program that will executes the task.

Initial chapters present the theoretical part related to the topic of work - such as *automats* and *transducers*. The following sections show the tools that were used during the construction of the program, along with their descriptions and examples of use.

The following chapters were written on the basis of own research. The raport on the application includes codes, descriptions of the algorithms that make up the program for automatic translation of texts and information on how to use this program. The application has been evaluated on the basis of the sample text using a quality measure.

Spis treści

Rozdział 1

Wprowadzenie

1.1. Teza pracy

Celem niniejszej pracy jest zweryfikowanie tezy, czy możliwy jest automatyczny przekład archaicznych tekstów języka polskiego na język współczesny.

1.2. Opis pracy

Świat, w którym współcześnie żyjemy, jest zbudowany na podstawie wiedzy zbieranej latami przez naszych przodków. Bez jej przekazywania rozwój cywilizacji stanąłby w miejscu. Wiedza ta, początkowo przekazywana słownie, została spisana na papierze, co z kolei przyczyniło się do rozkwitu nauki. Język, podobnie do żywego organizmu, z biegiem czasu ewoluuje, zmienia się pod wpływem czasu. Dla współczesnego, przeciętnego człowieka, język, który był używany 100, 200 lat wcześniej może być już niezrozumiały, a z kolei wiedza zawarta w starych księgach - bezużyteczna. Na świecie istnieją lingwiści, którzy interesują się ewolucją języka i teksty archaiczne nie stanowią dla nich problemu. Chcąc zautomatyzować proces przekładu tekstów na teksty współczesne powstał pomysł stwo-

rzenia programu komputerowego, który w oparciu o reguły dostarczone przez lingwistów wykona to zadanie.

Drugi rozdział, *Podstawy teoretyczne*, został podzielony na dwa podrozdziały: *Automaty* oraz *Transduktory*. Zostały w nim poruszone zagadnienia, które okazały się pomocne podczas budowy programu oraz reguł. W rozdziale trzecim omówione są narzędzia zastosowane w trakcie budowy programu: *OpenFST* oraz *Thrax*. Znajduje się tam krótki wstęp do każdej z technologii oraz przykładowe przypadki użycia – tworzenie i kompilacja transduktorów oraz automatów skończenie stanowych. Dalsza część pracy, czyli rozdziały czwarty i piąty, została napisana w oparciu o badania własne. Część *Projekt aplikacji* zawiera w sobie kody, opisy algorytmów składających się na program do automatycznego przekładu tekstów oraz informację, jak należy z tego programu korzystać. *Ewaluacja* natomiast skupia się na ocenie zastosowanego rozwiązania na podstawie przykładowego tekstu z użyciem miary jakości.

Rozdział 2

Podstawy teoretyczne

2.1. Automat

W okolicach lat czterdziestych i pięćdziesiątych XX w. niektórzy badacze zaczęli się zajmować maszynami, które są dzisiaj nazywane *automatami*. Miały one na celu modelowanie działania mózgu, ale okazały się użyteczne do innych celów, jak np.:

- skanowanie obszernych tekstów w celu znalezienia określonych słów,
- projektowanie obwodów cyfrowych oraz ich testowanie,
- analizatory leksykograficzne.

Automatem skończenie stanowym nad alfabetem A nazywamy system $A = (S, f)$, w którym:

- S - jest dowolnym skończonym zbiorem, zwanym zbiorem stanów,
- $f : S \times A \to S$ - jest funkcją przejść.

Automat będąc w stanie s_i po przeczytaniu litery a zmienia stan na s_j zgodnie z funkcją przejścia $f(s_i, a) = s_j$. Funkcję przejść rozszerzamy na cały wolny monoid $A*$ do

postaci $f : S \times A* \to S$, przyjmując: dla każdego $s \in S$ $f(s, 1) = s$ oraz dla każdego $s \in S$, $a \in A$ i dla dowolnego $w \in A*$ $f(s, wa) = f(f(s, w), a)$.

„Automaty skończone obejmują stany i przejścia między stanami w reakcji na wejścia.”[Hopcraft, 2005, s. 34]

Rysunek 2.1. Automat *wrak*

Źródło: Opracowanie własne

Rysunek 2.1. przedstawia automat skończenie stanowy, akceptujący napis *wrak*. Stan początkowy odpowiada łańcuchowi pustemu, z każdego stanu można przejść do kolejnego poprzez akceptację kolejnej litery wyrazu *wrak*. Do stanu o etykiecie *wrak* przechodzimy, kiedy automat rozpozna cały napis *wrak*.

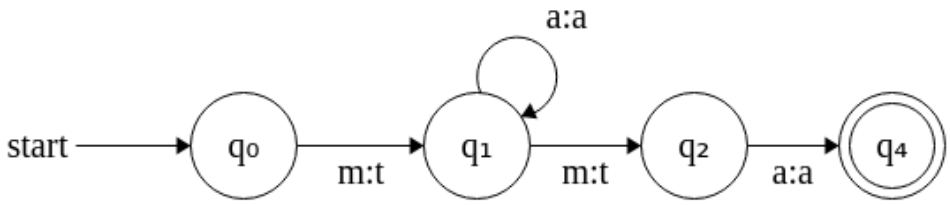
2.2. Transduktor

Transduktor to automat skończony z wyjściem. Elementem wejściowym jest tekst, który zostanie zmieniony według określonych zasad. Transduktor dla każdego symbolu z wejścia dopasowuje odpowiedni symbol dla wyjścia. Odgrywają one bardzo ważną rolę w zagadnieniach związanych z przetwarzaniem języka naturalnego. Transduktory zapisuje się formalnie w sposób przedstawiony poniżej:

$$T = (Q, \Sigma_1, \Sigma_2, \delta, \sigma, q_0, F)$$

- Q - skończony zbiór stanów,
- Σ_1 - skończony alfabet wejściowy,
- Σ_2 - skończony alfabet wyjściowy,
- $\delta(q, a)$ - funkcja przejścia, gdzie q należy do Q, a należy do Σ_1,
- $\sigma(q, a)$ - funkcja wyjściowa, która zwraca wynik $\sigma(q, a) = a$ należący do Σ_2,
- q_0 - stan początkowy należący do Q,
- F - zbiór stanów końcowych zawarty w Q.

Przykładowy transduktor został przedstawiony na Rysunku 2.2.



Rysunek 2.2. Transduktor *ma*ma*

Źródło: Opracowanie własne

Transduktor z Rysunku 2.2. na wejściu może przyjąć napis *mama*, który zostanie przekształcony w napis *tata*. Dla napisu *maaaama*, wyjściowym napisem będzie *taaata*.

Rozdział 3

Narzędzia do przetwarzania automatów

W tym rozdziale zostaną omówione główne narzędzia do przetwarzania automatów i reguł, które zostały wykorzystane w projekcie magisterskim.

3.1. OpenFST

Open FST to biblioteka stworzona do budowania, łączenia, optymalizacji oraz przeszukiwania skończenie stanowych transduktorów z wagami. Została ona napisana przez zespół instytutów badawczych: *Google Research* i *NYU's Courant Institute*. Zaprojektowana jest z myślą o przetwarzaniu dużej ilości danych. Jak podają autorzy, biblioteka jest wydajna, elastyczna i kompleksowa. Biblioteka została udostępniona na licencji *Apache*[1].

[1] *Licencja Apache* - licencja wolnego oprogramowania autorstwa Apache Software Foundation. Licencja ta dopuszcza użycie kodu źródłowego zarówno na potrzeby wolnego oprogramowania, jak i zamkniętego oprogramowania komercyjnego. Pod tą licencją rozpowszechniane jest oprogramowanie tworzone przez Apache Software Foundation.

Z biblioteki *Open FST* można korzystać na dwa sposoby:

- dołączając ją do swojego kodu:

```
#include <fst/fstlib.h>
using namespace fst;
```

Wydruk 3.1. Dołączenie biblioteki *Open FST* w języku *C++*

- tworząc plik tekstowy z regułami i wywołując odpowiednie komendy z linii poleceń w katalogu instalacyjnym, sposób ten został opisany poniżej, dla transduktora z Rysunku 3.1.

Rysunek 3.1. Transduktor 1

Źródło: Opracowanie własne

Wagi pozwalają okreslić, który stan jest bardziej odpowiedni dla danego napisu, a co za tym idzie stosowanie wag przyczynia się do tworzenia bardziej trafionych transduktorów. Na Rysunku 3.1. początkowym stanem jest stan *0*, a końcowym *2* z wagą 3.5. Początkowy stan może być tylko jeden. Dla ciągu *ak* powyższy transduktor zwróci nam ciąg *sm* z wagą 1.5+2.5+1+3.5=8.5.

Wydruk 3.2. przedstawia transduktor z Rysunku 3.1. napisany w języku *C++*:

```
//dodaje stan 0 do pustego FST i nadaje mu stan poczatkowy
fst.AddState();
fst.SetStart(0);
//do stanu 0 dodaje galaz
```

```
fst.AddArc(0,StdArc(1,1,1,1));
//dodaje ostatni - 2 stan, z waga i ustawiamy go jako stan koncowy
fst.AddState();
fst.SetFinal(2,3.5);
fst.Write("transduktor.fst");
```

Wydruk 3.2. Transduktor 1 napisany w języku *C++*

Transduktor z Wydruku 3.2. można napisać i skompilować za pomocą pliku tekstowego i linii poleceń, co przedstawia Wydruk 3.3.:

```
$cat>text.fst<<EOF
//wartosci oznaczaja kolejno: stan poczatkowy,galaz,zasada dzialania, waga
0 1 a s 1
1 2.5
1 2 k m 1
2 3.5
EOF
```

Wydruk 3.3. Transduktor 1 napisany w pliku tekstowym

Jeżeli chcemy oznaczać gałęzie etykietami, musimy stworzyć dwa pliki, które mapują liczby na symbole – Wydruk 3.4.

```
$cat>isymbols.txt<<EOF
<eps>0
a 1
k 2
EOF

$cat>osymbols.txt<<EOF
<eps>0
```

```
s 1
m 2
EOF
```

Wydruk 3.4. Transduktor 1 napisany w pliku tekstowym – mapowanie liczb na symbole

Wydruk 3.5. ukazuje kompilację, stworzonych wcześniej plików tekstowych (Wydruki 3.3.,3.4.) do pliku binarnego *fst*.

```
$cat>isymbols.txt<<EOF
<eps>0
a 1
k 2
EOF

$cat>osymbols.txt<<EOF
<eps>0
s 1
m 2
EOF
```

Wydruk 3.5. Kompilacja Transduktora 1, napisanego w pliku tekstowym.

Aby skorzystać z utworzonego transduktora w programie napisanym w języku *C++* należy użyć komendy z Wydruku 3.6.

```
StdFst *fst = StdFst::Read("transduktor.fst");
```

Wydruk 3.6. Wczytywanie transduktora w pliku *fst* w języku *C++*

Przykładowym transduktorem stworzonym z użyciem biblioteki *OpenFST* będzie transduktor z Wydruku 3.7. oraz Rysunku 3.2., którego zadaniem jest zamiana

wielkich liter A i B na litery alfabetu greckiego α i β.

```
0 0 A α
0 0 B β
0
EOF
```

Wydruk 3.7. Plik tekstowy – zamiana na litery greckie

Rysunek 3.2. Transduktor zamieniający litery na greckie

Źródło: Opracowanie własne

3.2. Thrax

Thrax to kompilator gramatyk używających wyrażenia regularne. Gramatyki przechowywane są w plikach *.grm*. *Thrax* kompiluje je do automatów skończonych, które stanowią format wejściowy do biblioteki *OpenFST*. Został on stworzony przez zespół w którego skład wchodzą: Terry Tai, Wojciech Skut and Richard Sproat, którzy pracują dla firmy *Google*. Narzędzie to jest udostępnione, tak jak w przypadku biblioteki *OpenFst*, na licencji *Apache*.

3.2.1. Funkcje i deklaracje

W formacie *Thrax* można stosować funkcje oraz deklaracje, które mają podobne zastosowanie do funkcji i deklaracji w językach programowania. Kod napisany staje się czytelniejszy, rozmiar kodu jest zmniejszony, czas potrzebny na napisanie transduktora maleje, kod łatwo poprawić oraz pomniejszone zostaje ryzyko błędów.

- Funkcje

  Na Wydruku 3.8. widnieje przykładowa funkcja.

```
func addMini[fst]{
   //dolaczamy przedrostek "mini"...
   result = "mini"fst;
   //zwracamy wyniki
   return result;
}
```

Wydruk 3.8. Funkcja *addMini* - *Thrax*

Funkcja z Wydruku 3.8. tworzy transduktor, który rozpoznaje ciągi utworzone przez dodanie przedrostka *mini* do wyrazów, które rozpoznaje transduktor. Jeśli np. trans-

duktor o nazwie *van* rozpoznaje wyraz *van*, to *addMini[van]* dodaje łańcuch *mini* do wyrazu *van* i powstawje wyraz *minivan*.

- Deklaracje

  Wydruk 3.9. przedstawia przykładową deklaracje w bibliotece *Thrax*.

```
slownie = "jeden";
export liczby = slownie|"1";
```

Wydruk 3.9. Przykladowa deklaracja *slownie* - *Thrax*

Transduktor o nazwie *slownie* z Wydruku 3.9., rozpoznaje napis *jeden* (i kopiuje na wyjście). Transduktor o nazwie *liczby* rozpoznaje dwa napisy (i kopiuje na wyjście). Do pliku *far* eksportowany jest tylko *fst* o nazwie *liczby* (stanowi o tym dyrektywa *export*). Transduktor *slownie* służy wyłącznie jako pomocniczy.

3.2.2. Oznaczenia w Thrax

Wyrażenia i oznaczenia w *Thrax* są kompilowane na transduktory i odpowiadające im operacje:
Wydruk 3.10. obrazuje zastosowania wyrażenia *, automat *o_zero_wiele* akceptuje znak *o* zero lub wiele razy.

```
o_zero_wiele = "o"*;
```

Wydruk 3.10. Oznaczenia w *Thrax* „*”

Wydruk 3.11. przedstawia zastosowanie znaku ?, automat *o_zero_jeden* akceptuje znak *o* zero lub jeden raz.

```
o_zero_jeden = "o"?;
```

Wydruk 3.11. Oznaczenia w *Thrax* „?”

Wydruk 3.12. wyjaśnia zastosowanie znaku +, transduktor *o_jeden_wiele* akceptuje znak *o* jeden lub wiele razy.

```
o_zero_jeden = "o"+;
```

Wydruk 3.12. Oznaczenia w *Thrax* „+”

Wydruk 3.13. obrazuje zastosowania wyrażenia {}, automat *o_n_k* akceptuje znak *o* od dwóch do sześciu razy.

```
o_n_k = "o"{2,6};
```

Wydruk 3.13. Oznaczenia w *Thrax* „{}”

Wydruk 3.14. przedstawia zastosowania wyrażenia @, składa ono automat *o_zero_wiele* z drugim automatem *o_n_k*.

```
o_zero_wiele @ o_n_k;
```

Wydruk 3.14. Oznaczenia w *Thrax* „@”

Wydruk 3.15. przedstawia zastosowanie wyrażenia :, transduktor *nie_tak* zamienia wejściowy łańcuch znaków *nie* na wyjściowy *tak*.

```
nie_tak = "nie":"tak";
```

Wydruk 3.15. Oznaczenia w *Thrax* „:”

Wydruk 3.15. obrazuje zastosowania wyrażenia <> - dodaje ono wagę dla łańcucha *aaa* o wartości 1.

```
o = "aaa"<1>;
```

Wydruk 3.16. Oznaczenia w *Thrax* „<>”

3.2.3. Kompilacja transduktora

Kompilację pliku tekstowego do formatu *far* przeprowadza się za pomocą komendy *thraxmakedep*. Jeden plik *grm* może posiadać wiele automatów, które mogą zostać skompilowane. Wyraz *export* odpowiada za wyeksportowanie automatu:

```
o_bez_kreski = "o";
o_z_kreska = "ó";
export Popraw = "m"o_bez_kreski"j":"m"o_z_kreska"j";
```

Wydruk 3.17. Kompilacja transduktora

W kodzie przedstawionym na Wydruku 3.17. jest wyeksportowany automat *Popraw* do pliku *popraw.far*.
Eksport do pliku *far* z pliku *grm* odbywa się za pomocą komendy przedstawionej na Wydruku 3.18.

```
$thraxmakedep --input_grammar=test.grm --output_far=test.far
```

Wydruk 3.18. Eksport pliku *grm* do pliku *far*.

3.2.4. Użycie stworzonych transduktorów

W celu użycia automatu w formacie *far* należy zastosować komendę z Wydruku 3.19.

```
$thraxrewrite-tester --far=test.far --rules=Popraw
Inputstring:[type your input here]
```

Wydruk 3.19. Użycie automatu z pliku *far*

Polecenie z Wydruku 3.19. ładuje plik *test.far* i jednocześnie umożliwiło testowanie transduktora *Popraw*, poprzez wpisanie do konsoli ciągu znaków (*Input string*).
Z pliku *far* możemy wyekstrahować poszczególne transduktory korzystając z komendy

widocznej na Wydruku 3.20.

```
$far2fst test.far Popraw
```

Wydruk 3.20. Wyekstrahowanie poszczególnych transduktorów

Po wyeksportowaniu jednego transduktora, można uzyskać informację o nim poprzez komendę znajdującą się w Wydruku 3.21.

```
$fstinfo Popraw.fst
```

Wydruk 3.21. Uzyskanie informacji o transduktorze

Rozdział 4

Projekt aplikacji

4.1. Założenia aplikacji

Celem projektu jest stworzenie narzędzia obsługiwanego z linii poleceń (nazwanego *wrapperem*) do przetwarzania tekstu napisanego archaicznym językiem polskim, które na podstawie dostarczonych reguł, zamienia nieużywane wyrazy na ich odpowiedniki w języku obecnym.
wrapper ma możliwość stworzenia pliku wyjątków – jest to lista słów w pliku *exception.txt* wygenerowana na podstawie słownika wyrazów *sjp*, na których zadziałały dostarczone reguły. Wyrazy z pliku wyjątków są wyłączone z przetwarzania podczas właściwej pracy programu.
Reguły są dostarczane w formacie zgodnym z *OpenFST*, *Thrax* w pliku *far* lub *grm*. Lista nazw reguł, które mają zostać wykorzystane, wczytywana jest z pliku *txt*.
Wynik przetwarzania tekstu jest widoczny w konsoli lub w pliku *xml* w formacie *JUnit*[1].
Narzędzie może posłużyć do przetwarzania dokumentów, ksiąg itp.

[1] *Junit* – framework do testów jednostkowych w języku *Java*.

4.2. Wymagania aplikacji

Program wymaga, aby środowiskiem uruchomienia był system *GNU/Linux*, dystrybucja *Ubuntu* wyposażona w narzędzia *gcc 4.8*, *OpenFST* z włączonymi opcjami *far, pdt, mpdt, ngram-fsts* oraz narzędziem *Thrax*. Wydruk 4.1. przedstawia poprawną komendę do konfiguracji biblioteki.

```
$./configure --enable-far=yes --enable-pdt=yes --enable-mpdt=yes
    --enable-ngram-fsts=yes
```

Wydruk 4.1. Konfiguracja *OpenFST*

Program zawiera odwołania do bibliotek (przedstawione na Wydruku 4.2.), więc muszą one być dostępne w systemie.

```
#include"./pstreams/pstream.h"
#include<string>
#include<iostream>
#include<string.h>
#include<stdio.h>
#include<sstream>
#include<string>
#include<fstream>
#include"Test.h"
#include<list>
#include<algorithm>
#include<utility>
#include<boost/algorithm/string.hpp>
```

Wydruk 4.2. Biblioteki użyte w programie *wrapper*

4.3. Implementacja algorytmów

Algorytm tworzenia wyjątków jest przedstawiony na Wydruku 4.3.

```
bool make_exception_file(std::string dictionaryFilePath, std::string
    farDirectory, std::vector<Test> allTest){
    std::string exceptionsList="";
    std::string wordList="";
    std::string filePath = "wrapper/"+dictionaryFilePath;
    std::string line;
    std::vector<std::string> contentVector;
    std::fstream fs (filePath.c_str(), std::fstream::in | std::fstream::out);
    int i=0;
    int interval=100;
    if (!fs) std::fstream fs (dictionaryFilePath.c_str(), std::fstream::in |
        std::fstream::out);
    while (std::getline(fs, line)){
        i++;
        std::cout<<i<<std::endl;
        std::string word = line;
        if ((unsigned int)word[word.size()-1]==13) word.erase(word.end()-1);
        wordList +=word+" ";
        if (i>=interval){
            wordList.erase(wordList.end()-1);
            exceptionsList=is_changed_by_any_rule_process_input_exceptions(wordList,
                allTest, farDirectory);
            contentVector.push_back(exceptionsList);
            i=0;
            wordList="";
        }
```

```
    }
    if (i!=0){
        wordList.erase(wordList.end()-1);
        exceptionsList=is_changed_by_any_rule_process_input_exceptions(wordList,
            allTest, farDirectory);
        contentVector.push_back(exceptionsList);
        i=0;
        wordList="";
    }
    fs.close();
    std::string exceptionFilePath="exceptions.txt";
    std::ofstream xmlOut(exceptionFilePath.c_str());
    if (xmlOut){
            int lengthContentVector = contentVector.size();
            for (int i=0;i<lengthContentVector;i++)
                xmlOut<<contentVector.at(i);
    }
    else
        return false;
    xmlOut.close();
    return true;
}
```

Wydruk 4.3. Funkcja *make_exception_file*

Funkcja *make_exception_file* z Wydruku 4.3. jest odpowiedzialna za stworzenie pliku wyjątków. Jako jej argumenty przekazujemy ścieżki do: pliku słownika, pliku reguł w formacie *far* oraz listy zawierającej obiekty klasy *Test*, która odpowiedzialna jest za przechowywanie kolejnych wyników testów/przetwarzania.

Funkcja *make_exception_file* odczytuje kolejne słowa z pliku wskazanego w zmiennej *dictio-*

naryFilePath, a następnie dodaje do wektora *contentVector* te słowa, na których zostały zastosowane dostarczone reguły, czyli wyjątki. Finalnie wszystkie znalezione słowa są zapisywane do pliku *exception.txt*.

Implementację algorytmu usuwania wyjątków z zdania ukazuje Wydruk 4.4.

```
std::vector<SentenceException> split_sentence_by_exceptions(std::string
    sentence, std::vector<std::string> exceptionList){
    std::vector<std::string> sentenceExplode = explode(sentence.c_str(), ' ');
    std::vector<SentenceException> out;
    std::string sentencePart="";
    int length = sentenceExplode.size();
    SentenceException singleGroup;
    singleGroup.sentence="";
    for (int i=0; i<length; i++){
        if (is_string_is_in_list(sentenceExplode.at(i),exceptionList)){
                if (!are_equal(singleGroup.sentence,"")){
                    singleGroup.isException=false;
                    out.push_back(singleGroup);
                }
                singleGroup.isException=true;
                singleGroup.sentence=sentenceExplode.at(i);
                out.push_back(singleGroup);
                singleGroup.sentence="";
            }
        else
             singleGroup.sentence+=sentenceExplode.at(i)+" ";
        if (i==length-1 && !are_equal(singleGroup.sentence,"")){
            singleGroup.isException=false;
            out.push_back(singleGroup);
        }
```

```
    }
    return out;
}
```

Wydruk 4.4. Funkcja *split_sentence_by_exception*

Funkcja *split_sentence_by_exception* (Wydruk 4.4.) jako argumenty przyjmuje kolejno: zdanie wejściowe *sentence* oraz wektor wyjątków *exceptionList*, a zwraca strukturę *SentenceException* (Wydruk 4.5.), która przechowuje tekst wejściowy w grupach oznaczonych jako wyjątki lub nie. Funkcja rozdziela wprowadzony łańcuch znaków na wyrazy i dla każdego wyrazu sprawdza, czy jest on wyjątkiem, a w przypadku pozytywnym dodaje go do wyjściowej zmiennej *out* z flagą *isException* o wartości *true*.

```
struct SentenceException{
   std::string sentence;
   bool isException;
   void print(){
       std::cout<<"sentence: " sentence<<" isException: "
          <<isException<<std::endl;
   }
};
```

Wydruk 4.5. Struktura *SentenceException*

Do tworzenia pliku *far* na podstawie pliku *grm* służy algorytm przedstawiony na Wyd. 4.6.

```
std::string create_far_from_grm(std::string grmDirectory){
    std::string direcotryName;
    size_t found;
    found=grmDirectory.find_last_of(".");
    direcotryName = grmDirectory.substr(0,found);
```

```
    std::string command = "thraxcompiler --input_grammar=" grmDirectory "
        --output_far=" direcotryName ".far";
    std::string buffer;
    redi::pstream proc(command, redi::pstreams::pstdout |
        redi::pstreams::pstdin | redi::pstreams::pstderr);
    while (proc >> buffer){
        if (strcmp(buffer.c_str(),"FATAL:")==0){
            std::cout<<"error while creating far"<<std::endl;
            return "error while creating far";
        }
    }
    return direcotryName ".far";
}
```

Wydruk 4.6. Funkcja *create_far_from_grm*

W przypadku podania na wejściu pliku *grm*, należy ten plik przetworzyć do pliku *far*, aby *Thrax* mógł posługiwać się nim. Proces ten przedstawiony jest na Wydruku 4.6.
Funkcja *create_far_from_grm* wywołuje komendę *thraxcompiler* (udostępnioną przez narzędzie *Thrax*), która przyjmuje jako argumenty:

- *input_grammar* – scieżkę do pliku *grm*,
- *outpur_far* – ścieżkę do wyjściowego pliku *far*.

Jeżeli komenda zostanie wykonana poprawnie, to funkcja *create_far_from_grm* zwróci ścieżkę do utworzonego pliku *far*.

Główny algorytm, czyli algorytm przetwarzania tekstu wejściowego przedstawia Wydruk 4.7.

```
void process_input(std::vector<SentenceException> input, std::vector<Test>
    allTest, std::string farDirectory, int outputFormat=1, bool debug=false,
    std::string xmlFileInPath=""){
    int numberOfGroupsInSentence = input.size();
    std::string getherOut = "";
    std::string inputSentence="";
    std::string getherIn ="";
    for (int b=0;b<numberOfGroupsInSentence;b ){
        getherIn =input.at(b).sentence;
        inputSentence=input.at(b).sentence;
        if (input.at(b).isException==true){
            getherIn =" ";
            getherOut =input.at(b).sentence " ";
            continue;
        }
        std::string inputString = input.at(b).sentence;
        std::string parseWord;
        int allTestSize=allTest.size();
        for (int i=0;i<allTestSize;i ){
            allTest.at(i).sentenceToParse=boost::trim_copy(inputString);
            boost::trim(inputString);
            std::string rules = allTest.at(i).rule;
            std::string command = "thraxrewrite-tester --far=" farDirectory "
                --rules=" rules;
            std::string thraxTesterOutput ="";
            parseWord = "";
            std::string buffer;
```

```
        redi::pstream proc(command, redi::pstreams::pstdout |
            redi::pstreams::pstdin | redi::pstreams::pstderr);
        while (proc >> buffer)
            if (strcmp(buffer.c_str(),"string:")==0) break;
        proc << inputString<<std::endl;
        bool checkOutput=false;
        while (proc >> buffer){
            thraxTesterOutput =buffer " ";
            if (checkOutput){
                if (strcmp(buffer.c_str(),"Input")==0) break;
                parseWord = buffer " ";
            }
            if (strcmp(thraxTesterOutput.c_str(),"Output string: ")==0)
                checkOutput=true;
            if (strcmp(thraxTesterOutput.c_str(),"Rewrite failed. ")==0)
                break;
        }
        if (parseWord.length()>0){
            parseWord.erase(parseWord.end()-1);
            inputString=parseWord;
        }
        allTest.at(i).isProcessed=true;
        allTest.at(i).result=parseWord;
        if (debug && !are_equal("", allTest.at(i).result) &&
            !are_equal(parseWord,allTest.at(i).sentenceToParse))
            allTest.at(i).print();
    }
    if (inputSentence.compare(inputString)==0) inputString="";
    if (are_equal(inputString,""))getherOut =inputSentence;
```

```
        else
            getherOut =inputString " ";
    }
    std::cout<<"Wyjscie: " getherOut<<std::endl;
    if (outputFormat!=3) parse_thrax_tester_output(outputFormat, getherOut
        ,getherIn, xmlFileInPath);
}
```

Wydruk 4.7. Funkcja *process_input*

Główną funkcją do przetwarzania pliku wejściowego jest funkcja *process_input* przedstawiona na Wydruku 4.7., która przyjmuje jako argumenty:

- *input* – wektor wyjątków stworzony dzięki funkcji,
- *allTest* – wektor kolejnych reguł do przetworzenia,
- *farDirectory* – ścieżkę do pliku *far*,
- *outputFormat* – format wyjściowy wyników (1 dla wyjścia w formacie pliku *xml*, 2 dla wyświetlenia wyjścia w linii komend),
- *debug* – flaga, dzięki której użytkownik dostaje więcej informacji na temat przeprowadzanego procesu,
- *xmlFileInPath*- ścieżka do pliku z tekstem do przetworzenia, w przypadku przetwarzania pliku *xml*.

Funkcja *process_input* przetwarza kolejno każdą grupę zdań ze zmiennej *input*. W przypadku napotkania wyjątku instrukcje znajdujące się w iteracji są omijane, a w przypadku grupy zwykłej, stosowane są kolejno reguły dodane do zmiennej *allTest*. Reguły przetwarzane są poleceniem *thraxrewrite-tester*, któremu należy przekazać plik *far* oraz nazwę

reguły. Tekst jest przesyłany do otwartego strumienia *proc*, a jego wyjście jest przypisywane do zmiennej *getherOut*, której wartość później jest porównywana z grupą wejściową. Jeżeli wartości są różne oznacza to, że reguła została zastosowana.

4.4. Obsługa aplikacji

Aby rozpocząć pracę z aplikacją należy skompilować kod *C++*. Plik wyjściowy kompilacji możemy uruchomić z argumentami:

- *grm [sciezka]* - ścieżka do *grm*,
- *far [sciezka]* - ścieżka do pliku *far*,
- *sentence "[zdanie]"* - zdanie do przetworzenia; jeżeli argument nie jest podamy, to zdanie wpisujemy na standardowym wejściu (aby wyjść z pętli trzeba podać napis *exit*),
- *rulesfile [sciezka]* - ścieżka do pliku z nazwami reguł do przetworzenia,
- *xml [sciezkaDoPlikuIn]* - wyjściem staje się plik *nazwa.out* (format *XML JUnit*) w katalogu pliku *nazwa.in*, na podstawie pliku *nazwa.exp* (oczekiwane wyjście),
- *debug* - dzięki tej fladze na standardowym wyjściu pojawiać się będzie wejściowy łańcuch znaków kolejno przetwarzany przez reguły z pliku *rulesfile*,
- *help* - wyświetla pomoc,
- *makeexception* - tworzy plik *exceptions.txt* w katalogu *wrapper*, na podstawie pliku *dictionary.txt*.

Wydruki 4.8., 4.9. oraz 4.10. przedstawiają przykładowe użycie aplikacji.

```
./wrapper.out -grm /home/patryk/test.grm -rulesfile /home/patryk/test.txt
    -sentence "zdanie zdanie"
```

Wydruk 4.8. Użycie programu *wrapper* – komunikacja z linii komend

Wydruk 4.8. przedstawia użycie programu *wrapper*, podając zdanie do przetworzenia, jako parametr *sentence*.

Komenda służąca do stworzenia pliku *exceptions* została przedstawiona na Wydruku 4.9.

```
./wrapper.out -far /home/patryk/test.far -rulesfile /home/patryk/test.txt
    -makeexception
```

Wydruk 4.9. Użycie programu *wrapper* – stworzenie pliku exception

Uruchomienie programu *wrapper* z zapisem do pliku *xml* ukazuje Wydruk 4.10.

```
./wrapper.out -far /home/patryk/test.far -rulesfile /home/patryk/test.txt -xml
    /home/patryk/wejscie.in
```

Wydruk 4.10. Użycie programu *wrapper* – wyjście do pliku *xml*

Rozdział 5

Ewaluacja

5.1. Miara ewaluacji

W celi sprawdzenia poprawności zaprojektowanego rozwiązania wykorzystana została miara jakości, która jest stosowana podczas oceny jakości i klasyfikacji. Na Rysunku 5.1. przedstawione są pojęcia związane z miarą ewaluacji:

- *klasa negatywna* – klasa, składająca się z elementów, na których nie zostały zastosowane operacje / reguły,
- *klasa pozytywna* – klasa, składająca się z elementów, na których zostały zastosowane operacje / reguły,
- *false negatives* – elementy niepoprawnie oznaczone jako elementy *klasy negatywnej*,
- *true negatives* – elementy poprawnie oznaczone jako elementy *klasy negatywnej*,
- *true positives* – elementy poprawnie oznaczone jako elementy *klasy pozytywnej*,
- *false positives* – elementy niepoprawnie oznaczone jako elementy *klasy pozytywnej*,

- *precision* – jest to liczba określająca precyzję rozwiązania, im większa precyzja, tym mniejsze jest ryzyko błędnego zakwalifikowania elementu *klasy pozytywnej*,

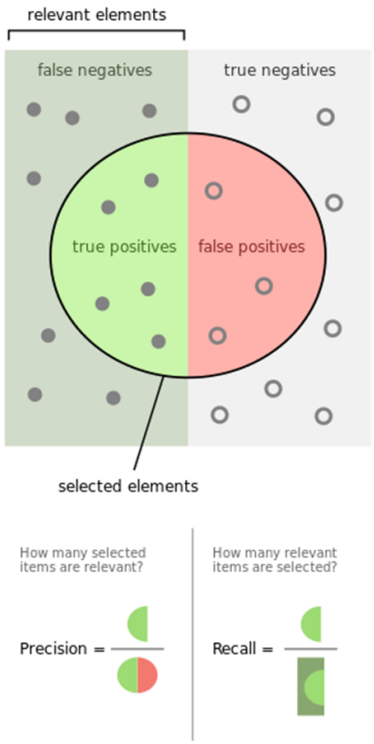
$$precision = \frac{|true \quad positives|}{|true \quad positives| + |false \quad negatives|}$$

- *recall* – jest to liczba, która mówi w jakim stopniu na elementach zostaną przeprowadzone operacje/reguły. Zwiększając wartość *recall* spada wartość *precision*, gdyż reguły zostaną zastosowane na większej ilości elementów (również na tych, na których nie powinny być zastosowane),

$$recall = \frac{|true \quad positives|}{|true \quad positives| + |false \quad positives|}$$

- *f-measure* - miara dokładności rozwiązania.

$$fmeasure = 2 * \frac{precision * recall}{precision + recall}$$



Rysunek 5.1. Miara ewaluacji

Źródło: www.wikipedia.pl

Wartości miary ewaluacji dla Rysunku 5.1. przedstawia Wydruk 5.1.

```
suma = 22
truePositive = 5
trueNegative = 3
falsePositive = 7
falseNegative = 7
precision ~ 0,4167
recall ~ 0,4167
f-mesure ~ 0,4167
```

Wydruk 5.1. Przykładowa miara ewaluacji

W odniesieniu do tematu pracy powyższe pojęcia oznaczają:

- *true positive* – oznaczać będzie wyraz zmieniony na poprawny,
- *true negative* – wyraz, który nie powinien zostać zmieniony i nie został zmieniony,
- *false positive* – wyraz, który został zmieniony, a nie powinien,
- *false negative* – wyraz, który nie został zmieniony, a powinien,
- *precision* – wartość predykcyjna dodatnia: liczba wyrazów typu *true positive* podzielona przez sumę wyrazów *true positive* oraz *false positives*,
- *recall* – liczba wyrazów typu *true positive* podzielona przez sumę *false negatives* oraz *true positives* i *false negatives*,
- *f -mesure* – miara dokładności rozwiązania.

5.2. Wyniki

Do oceny jakości rozwiązania wykorzystane zostały:

- tekst wejściowy - znajdujący się w Dodatku A.0.1.,
- tekst wejściowy - znajdujący się w Dodatku A.0.2.,
- zbiór reguł, który zawiera plik z Dodatku A.0.3. - został on opracowany przez promotora pracy.

Tekst wejściowy oraz reguły zostały wprowadzone do programu jako argumenty. Po przetworzeniu tekstu wyjście zostało porównane z oczekiwanym wyjściem. Na Wydruku 5.2. znajdują się dane zebrane podczas procesu oceny jakości. Dane te zostały zgromadzone, a poszczególne miary zostały obliczone manualnie.

```
sumaSlow = 1869
truePositive = 110
trueNegative = 1761
falsePositive = 3
falseNegative = 22
precision = truePositive / (truePositive + falsePositive) ~ 0,9734513274336283
recall = truePositive / (truePositive + falseNegative) ~ 0,8333333333333333
f-mesure = 2* (precision * recall) / (precision + recall) ~ 0,89795918367346
```

Wydruk 5.2. Dane – ewaluacja

5.3. Ocena wyników

Na podstawie wyników z Wydruku 5.2. można ocenić, że program *wrapper* działa w sposób poprawny, ale nie doskonały. Pomiędzy miarami *precision* oraz *recall* są zachowane proporcje na tyle odpowiednie, że znalazłby on zastosowanie przy małych niekomercyjnych projektach. W programie *wrapper* ważniejsza jest miara *precision* niż *recall*, więc należałoby dążyć do jej poprawy. Na podstawie analizy błędów na Wydruku 5.3. możemy stwierdzić, że algorytm *split_sentence_by_exceptions* wymaga poprawy, dzieli on segmenty wyrazów w nieodpowiednich miejscach, przez co nie mogą być na nich zastosowane odpowiednie reguły.

```
in = iakiemi są;
out = jakimi są;
expected = jakimi zą;
```

Wydruk 5.3. Błędy w programie *wrapper*

Natomiast błędy przedstawione na Wydruku 5.4., wskazują na to, że należałoby udo-

skonalić reguły.

```
in = LX.
out = LX.
expected = LKs.
```

Wydruk 5.4. Błedy w regułach, program *wrapper*

Wyniki te są subiektywne, ponieważ nie zostały znalezione inne rozwiązania, zajmujące się tym samym problemem.

5.4. Dalszy rozwój

Program *wrapper* ma szerokie pole do rozwoju. Pierwszym krokiem w tym kierunku byłoby przyśpieszenie wykonywania programu, np. algorytm *process_input* powinien korzystać z biblioteki *Thrax*, a nie wywoływać program *thraxrewrite-tester* z linii komend, na pewno usprawniłoby to drastycznie działanie programu. W ten sposób program mógłby zostać wykorzystany do przetwarzania dużej ilości danych.
Kolejną możliwością rozwoju programu byłoby wprowadzenie możliwości dodawania grup reguł z przypisanym do nich przedziałem lat, w których takie zasady językowe miały zastosowanie. Pozwoliłoby to na stworzenie programu do rozpoznawania lat, w których powstał dany tekst.
Przydatnym rozwiązaniem mogłaby okazać się rozbudowa programu o system *OCR* [1]. Wówczas wprowadzałoby się do programu już nie tylko tekst, ale bezpośrednio zdjęcie strony z tekstem. Działałoby to jak swego rodzaju translator i mogłoby okazać się bardzo przydatne dla osób pracujących w bibliotekach czy też archiwach z tekstem archaicznym.

[1] *OCR* - (ang. Optical Character Recognition) – zestaw technik lub oprogramowanie służące do rozpoznawania znaków i całych tekstów w pliku graficznym o postaci rastrowej. Zadaniem *OCR* jest zwykle rozpoznanie tekstu w zeskanowanym dokumencie (np. papierowym formularzu lub stronie książki). Źródło: http://pl.wikipedia.org

Rozdział 6

Podsumowanie

Problem czytania tekstów archaicznych, dotyczy nas i dotyczyć będzie naszych potomków, dla których współczesny język stanie się niezrozumiały. Tłumaczenie tekstów starodawnych będzie cały czas (patrząc na historię) problemem aktualnym.
Praca w swoim założeniu miała zająć się procesem automatycznego przekładu tekstów archaicznych na teksty współczesne w oparciu o reguły językowe. Problematyczny cały czas jest wątek wyjątków, który na pewno niesie ze sobą ryzyko błędnego oznaczenia przynależności wyrazów. Uważam, na podstawie otrzymanego wyniku *f-mesure* (na poziomie 0,89795918367346), że udało się stworzyć program, który w stopniu zadowalającym realizuje stawiane przed nim zadanie. Jest to ocena subiektywna, ponieważ nie zostały znalezione inne prace/ programy wykonujące taki proces. Fakt, że prawdopodobnie jest to pierwsze rozwiązanie tego typu może czynić ten temat jeszcze bardziej atrakcyjnym. Program *wrapper* jest dobrym rdzeniem do stworzenia bardziej zaawansowanego i szybszego rozwiązania.

Bibliografia

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

[2] OpenFST.
`http://openfst.cs.nyu.edu/twiki/bin/view/FST/WebHome`

[3] Thrax.
`http://openfst.cs.nyu.edu/twiki/bin/view/GRM/Thrax`

Dodatek A

Dodatki

A.0.1. Tekst wejściowy

Stawianie arkady . . . rusztowania.

54. Gdy wszystkie części arkady, będąc jak énajlepij z sobą połączone, stanowią jednę całość; wypada ztąd, że dla utrzymania éjj w czasie roboty, nie ma potrzeby wznoszenia ciągłych, w łuki prowadzonych rusztowań, czyli bukszteli, jak pod sklepienia kamienne; lecz dosyć jest éoprzć ją w niewielu punktach na niewzruszonych podporach, między któremi można bez wielkich trudności składać zworniki, przy pomocy rusztowań wiszących (jak niżej się okaże), utrzymanych przez ukończone już części arkady, która tym sposobem sama sobie służyłaby za główne rusztowanie. Tak zaczynając od przyczołków, z któremi taż arkada jest mocno połączona, należy, w miarę postępowania roboty, podpierać ją w punktach, z początku bliższych A, B, C, (fig. 14), następnie zaś w punktach D i E, na taką jeden od drugiego odległość, na jaką części arkady CD, DE, bez niebezpieczeństwa utrzymać się mogą. Gdy w samym środku czyli w wierzchołku arkady, ma się kończyć

składanie étjże, w étm miejscu ézatm nie ma potrzeby stawiania énowj podpory, i podpora EI, od odpowiadającj sobie w édrugij połowie arkady, może być dwa razy więécj oddalona, niż od podpory DK. Odległość punktów podparcia może być jeszcze zmniejszona za pomocą wsporów FG, F'G', któremi arkada w punktach G,G'.. zostanie utrzymana. Tym sposobem największa długość, na jaką taż arkada bez podparcia rozciągać się będzie, nie dochodzi 170 st. (49 m.). Ciężar części G'G (nie licząc łuków bocznych, które édopiro po spocznieniu arkady na punkcie G, założone być winny), wynosi najwięécj 7000 cet. (280000 kil.); który na końcu étjże części sprawi skutek wyrównywający tylko połowie tego, czyli 3500 cet. (140000 kil.): moc zaś arkady jest taka, że będąc w punkcie G' podpartą, utrzymać może w końcu G około 8000 cetn, (320000 kil.) z zupełénm bezpieczeństwem (*) {(*) Moc arkady można porównać z mocą rury ékwadratowj étj ésamj szerokości i grubości, oraz tę samą ilość żelaza w przecięciu mającj; do znalezienia ézatm ciężaru, jaki takowa w końcu G utrzyma, potrzeba użyć wzoru Nru 24, w którym położywszy b=60.12, g=1/2, c=170, otrzymamy P=8130 cet.}; ézatm oprócz ciężaru własnego, jeszcze 4500 cet. obcego ciężaru ézniść jest w stanie.

Skład podpór.

55. Każda podpora ma podobieństwo do zwyczajnych kozłów używanych w rusztowaniach. Większe z nich EI, DK, złożone są z czterech, mniejsze zaś CL, BM, z dwóch tylko nóg; jak to fig. 14 w planie wyobraża. Figura 16 przedstawia Widok, w kierunku mostu, wielkiego kozła DI, którego nogi zajmują w podstawie przestrzeń około 160 stóp (46 m.) długą i 110 st. (32 m.) szeroką, w celu nadania temuż épotrzebnj stałości; u góry zaś zbliżone są ku sobie i połączone, tak u wierzchu, jako i w pośrodku éswj wysokości, krzyżami ukośnemi i belkami poziomemi, które zarazem mają służyć za

rusztowania do umieszczenia na nich potrzebnych kołowrotów czyli wind, i składania ciężarów.

56. Każda noga, któérj dolną część, fig. 17 w widoku i przecięciu, figura zaś 18 w planie wyobraża, ma kształt ostrokręgu ściętego, nieco pochyło wygiętego, około 30 st. (8,6 m.) średnicy w podstawie trzymającego, oraz wydrążonego wewnątrz; i złożona jest z belek albo éraczj z półczyzn czyli bali 6calowych (0,14 m.), (większa bowiem grubość nie jest potrzebna), stawianych jednych na drugich i z sobą stykających się, a éprzytm tak składanych, aby jedne pomiędzy drugie wchodziły i tym sposobem wiązały się z sobą. W tym celu najprzód wbite mają być pale ABC (fig. 13) 12 cali (0,29 m.) w kw. grube, stanowiące podstawę nogi, z których po dwa naprzemian jedne w wysokości D, drugie w E, poucinane zostaną. Na nich stawiane będą bale DF, EG..., tak aby podobnież naprzemian jedne wyżéj nad drugie wystawały, co w całéj długości nogi ma się powtarzać. Ażeby te wszystkie części stanowiły całość, nie podlegającą zmianie kształtu, utrzymane zostaną wewnątrz za pomocą kół, w miejscach DD, EE, FF, GG,... co 20 st. (5,8 m.) jedno od drugiego przypadających, do których końce bali mocnemi gwoździami przybite będą. W składaniu kół postawi się naprzód słup środkowy SS, dla utrzymania belek krzyżowych HI, KL, z któremi łączą się ramiona albo promienie SM, SM,.. Wszystkie promienie przedłużone są na kilka stóp dla zrobienia na nich rusztowań zewnętrznych, wewnątrz zaś przy obwodzie połączone wieńcem NOP, zbitym z bali, podobnie jak w kołach wodnych lub zwyczajnych buksztelach. Za każdem postawieniem w ścianie jednego rzędu bali, powinno być złożone takie koło w końcach tychże n. p. G, G..., dla zrobienia rusztowania RR, z któregoby nowy rzęd bali pomiędzy pierwszemi ustawić można było. {Wszystkie te bale będą jeszcze, wpośrodku éswj długości pomiędzy dwoma kołami, łączone z sobą żelaznemi klamrami k, k; przestwory zaś między temiż balami, gdzie potrzeba, wypełniane listwami lub klinami.

Potrzebne pochylenie nogi kozła można łatwo otrzymać przez wbijanie, gdzie należy, klinów pomiędzy stykające się końce bali. - Cała zresztą robota jest nader prosta i nie trudniejsza od innych w zwyczajnych konstrukcyach, jak tylko przez to, że na większą wysokość ciężary podnoszone być muszą, co, jak niżéj się okaże, przy użyciu wind jest mało znaczącą rzeczą.

Moc podpór czyli kozłów.

57. Podług doświadczeń Rondeleta do zgniecenia pręta krótkiego sosnowego 1 cal (0,024 m.) grubości w kwadrat mającego, potrzeba około 70 cetn. (2 800 kil.) (*). {(*){{Art de âbtir.- Navier: Application de la éMcanique etc.}}} W konstrukcyach jednak, takich nawet w których drzewo énajbardzij bywa obciążane (jak n.p. w palach fundamentowych), rzadko więécj liczy się na moc jednego cala kw. jak 6 cet. (0,4 kil. na 1 mm. kw.). - Kiedy długość pręta drewnianego lub belki, około 20 do 24 razy większa jest od jego grubości, wówczas, podług doświadczeń tegoż Rondeleta, moc drzewa pionowo obciążonego prawie dwa razy się zmniejsza. {W nogach kozłów opisanych takiż jest właśnie stosunek długości (10 st. = 2,88 m.), na jaką bale, między kołami i klamrami, bez utrzymania zostają, do grubości tychże bali (6 c.=0,144 m.). Moc ézatm onych wyniesie 3 cet. na 1 cal kw., albo około 400 cet. na 1 st. kw. (200 000 kil. na 1 m. kw.). Gdy powierzchnia przecięcia bali, składających niższą część nogi, énajmnij 45 st. kw. (3,6 m. kw.) czyni, na całą przeto moc éjednj nogi albo ciężar jaki takowa może dźwigać, wypada 18000 cet. (720 000 k.), czterech zaś nóg albo jednego kozła (72 000 cet. (2 880 000 kil.)

58. Największy ciężar części arkady spoczywającéj na jednym koźle, wynosić może wraz z rusztowaniami wiszącemi . . . . 30 000 cet. (1 200 000 kil.).

Ciężar kozła i rusztowań do niego należących . . 40 000 cet. (1 600 000 kil.).

Ogół ciężaru w podstawie kozła, wynosi . . . . 70 000 cet. (2 800 000 kil.); co prawie wyrównywa powyższj mocy (*) {(*) U wierzchu kozła, ciśnienie zmniejszone jest éprzynajmnij o połowę, o tyle étż ézatm zmniejszy się i ilość drzewa, oraz średnica nóg, która w étm miejscu wynosi 15 stóp.}. ROZDZIAŁ IX.

O TRACHYTACH I BASALTACH.

Dawne te Lawy wulkanicznych roztopionych skał, stanowią osobną i obszerną dzielnicę niepodległą ({{terrains éindpendans}}) skał zagadliwych, ({{problematiques}}), iakiemi są Trachyty i Basalty ; massy lite z feldspatu i Hornblendy, (rogomienia) albo z feldspatu i Augitu (1) {(1) Augit, ({{èPyroxne}}) nazwany tak od blasku; więc spolszczyć go można na blasczeń.} złożone, które Cordier nazwał skałami ogniorodnemi ({{roches èpyrognes}}), a które przez niego i przez Fleriau de Bellevue oryktognostycznie rozkładane (2) {(2) {{Journal de Physique}} Tom LI. LX. i LXXIII.}, dotąd w sposobie powstania swoiego nie są docieczone. Liczą się one do utworów wulkanicznych, bo neptunicznych pochód cale wyłączaią, ale żadnego niemaią podobieństwa do Law teraźnieyszemi gorzelcami wymiatanych.

Trachyty są porfyrowatemi skałami częstokroć dziurkowatemi, chropowatemi na dotknięcie i przeto tak przez Hauy nazwane, Trachys, szorstki, nierówny. (1) {(1) Można go nazwać szorścień.}

Dolomieu policzył ie do granitowych i porfyrowatych Law, a Buch nazwał ie Trappo Porfyrami. Są one albo same albo złączone z innemi oryktognostycznie różnemi skałami, iakoto Perlity, Obsydiany, gębczenie

(2) {(2) Gębczeń, pumex, pumite, Bimstein nie iest żadnym osobnym rodzajem skały, ale belkowatym, włóknistym stanem w który pewne wulkaniczne skały przechodzą; Hauy odróżnił i w Klassyfikacyi swoiey położył Trachyt, Perlit i Obsydian, jako sposobne do tego stanu przechodzić.} i inne dotąd nieopisane. Nigdzie ich niema w prądach iak nowe Lawy, ale po wszystkich częściach ziemi wznoszą się to w grona górzyn, iak Wygorzelec w Węgrzech (Vihoret) od 15 do 20 mil wzdłuż; to W całe pasma lub obszerne płyty ciągłey i potężney massy litey, iak w płycie Quito od 14 do 18 tysięcy stóp grubości karni. (1) {(1) Karnią górnicy nasi nazywaią grubość ławic, czyli złog rudy.} albo ogromnych rumowisk i zlepów częstokroć nad bliższe górzyny wzniośleyszych, iak W Węgrzech grzbiet między Neusohl i Kremnitz na 3186, i całe pasmo Matra na 2800 stóp wysokie.

Basalty, {{Basanites}}, starożytnych, są rodzaiem skał czarnych mianych za krzemień mniey wiecey z Rogomieniem (hornblendą) lub z Augitem zmieszany, i uważanych za zbitą rozmaitość Zielonka, (ügrnstein) który cześcią w Dyabasy i w Trappicie Brogniarta, częścią w Diorycie i Afanicie Hauyego zawiera się. Dla tego że skały te łatwo z innemi mieszaią się i trudne są do rozeznania, Brogniart nazwał ie {{mimose}} od {{mimos}} naśladowanie, o Hauay Dolerit od {{dorlos}} obłuda; niepodobna zaś częstokroć iest rozpoznać, czyli ta massa czarna iest rogomieniem (Hornblendą) lub Augitem. Co Basalty szczególnie od wszystkich innych skał, a mianowicie od Zielonka i Trachytu w składzie swoim odróżnia, iest Oliwień (olivin) którego w tamtych skałach cale niema; Augit zaś i magnesowe żelazo we wszystkich znachodzone bywa. Niema też nigdy w basaltowych złogach ani gębczenia ani Obsydianu, ani właściwego Trachytu; owszem obiedwie te skały zdaią się na wzaiem odpychać, nigdzie się niełączą, wszędy osobno stoią, a gdzie są sobie naybliżey, tam Basalt na Trachycie leży lub od niego zlepami gębczenia albo Trachytu odłączony. Często zaś Basalt w pierwotnych

skałach widziany starszym bydź może od niektórych Trachytów. Nakoniec, to ieszcze Basalty różni od Trachytów, iż się w prądach iak teraznieysze Lawy znayduią; tak mianowicie w Montpezat, gdzie zapełniaiąc dolinę padły, iak się wydaie na kałuże i nagłem parowaniem wody utworzyły sklepiste podniesienia mieyscami w słupy połupane; za tym widocznym prądem dochodzi się osobno stoiącego wzgórza, którego spusty kulastemi zuzlami są okryte, a na wierzchołku widać ostatki doskonałego Krateru. (1) {(1) {{Faujas de St Fond: Sur les Volcans eteints du Vivarais.}}} Lubo zaś nie wszędy tak wyraźne są basaltowe prądy, zawdy iednak te same na sobie noszą znamiona.

Długo czarne i zielone głazy pomników i obelisków egipskich uchodziły, zwłaszcza u Antykwaryuszów, za Basalty. Mineralogowie, i Werner sam, uwiedziony złomkami, miał ie za Syenit; nowsze śledzenia P. Wad na naypięknieyszym w tym względzie obelisku w Rzymie na Piazza Navona znayduiącym się, i P. èRozires w okolicach Syeny, na półwyspie góry Synai i w kamienistey Arabyi dowiodły: że skały te, z których podług wszelkiego podobieństwa Tablice prawa Moyżeszowe zrobione były i iak wieść niesie w teraźnieyszym Dziebel-Musa maią bydź zakopane, są przechodnim tworem granitycznym z rogomienia (Hornblendy) i krzemienia, z nieco kwarcu i błyszczu wmieszanego złożonym i dokładnie Synaytem nazwać się mogącym.

Wulkaniczne te utwory rozmaity wiek powstania swego wyraznie okazuią, i wszędy do przechodnich skał należeć się zdaią; to ie zagadliwemi czyni i nasuwa wniosek powolnego przeięcia od dołu do góry, oraz ścisły związek między naynowszemi złogami przechodniemi, naydawnieyszym osadem i dawnemi wulkanicznemi tworami. Gdy zaś Trachyt sam rzadko wolno na świat wychodzi i częstokroć pokryty iest piasczeniem i nowemi wapieniami, które znowu pokryte są wapieniem ieziernym z małzami; stąd poznieyszy iego wiek i porę powstania między tworem osadu i tak zwanemi trzeciakiemi gruntami

wniesiono.

A.0.2. Tekst wyjściowy

Stawianie arkady . . . rusztowania.

54. Gdy wszystkie części arkady, będąc jak najlepiej z sobą połączone, stanowią jednę całość; wypada stąd, że dla utrzymania jej w czasie roboty, nie ma potrzeby wznoszenia ciągłych, w łuki prowadzonych rusztowań, czyli bukszteli, jak pod sklepienia kamienne; lecz dosyć jest oprzeć ją w niewielu punktach na niewzruszonych podporach, między którymi można bez wielkich trudności składać zworniki, przy pomocy rusztowań wiszących (jak niżej się okaże), utrzymanych przez ukończone już części arkady, która tym sposobem sama sobie złużyłaby za główne rusztowanie. Tak zaczynając od przyczołków, z którymi taż arkada jest mocno połączona, należy, w miarę postępowania roboty, podpierać ją w punktach, z początku bliższych A, B, C, (fig. 14), następnie zaś w punktach D i E, na taką jeden od drugiego odległość, na jaką części arkady CD, DE, bez niebezpieczeństwa utrzymać się mogą. Gdy w samym środku czyli w wierzchołku arkady, ma się kończyć składanie tejże, w tem miejscu zatem nie ma potrzeby stawiania nowej podpory, i podpora EI, od odpowiadającej sobie w drugiej połowie arkady, może być dwa razy więcej oddalona, niż od podpory DK. Odległość punktów podparcia może być jeszcze zmniejszona za pomocą wsporów FG, F'G', którymi arkada w punktach G,G'.. zostanie utrzymana. Tym sposobem największa długość, na jaką taż arkada bez podparcia rozciągać się będzie, nie dochodzi 170 st. (49 m.). Ciężar części G'G (nie licząc łuków bocznych, które dopiero po spocznieniu arkady na punkcie G, założone być winny), wynosi najwięcej 7000 cet. (280000 kil.); który na końcu tejże części

sprawi skutek wyrównywający tylko połowie tego, czyli 3500 cet. (140000 kil.): moc zaś arkady jest taka, że będąc w punkcie G' podpartą, utrzymać może w końcu G około 8000 cetn, (320000 kil.) z zupełnem bezpieczeństwem (*) {(*) Moc arkady można porównać z mocą rury kwadratowej tej samej szerokości i grubości, oraz tę samą ilość żelaza w przecięciu mającej; do znalezienia zatem ciężaru, jaki takowa w końcu G utrzyma, potrzeba użyć wzoru Nru 24, w którym położywszy b=60.12, g=1/2, c=170, otrzymamy P=8130 cet.}; zatem oprócz ciężaru własnego, jeszcze 4500 cet. obcego ciężaru znieść jest w stanie.

Skład podpór.

55. Każda podpora ma podobieństwo do zwyczajnych kozłów używanych w rusztowaniach. Większe z nich EI, DK, złożone zą z czterech, mniejsze zaś CL, BM, z dwóch tylko nóg; jak to fig. 14 w planie wiobraża. Figura 16 przedstawia Widok, w kierunku mostu, wielkiego kozła DI, którego nogi zajmują w podstawie przestrzeń około 160 stóp (46 m.) długą i 110 st. (32 m.) szeroką, w celu nadania temuż potrzebnej stałości; u góry zaś zbliżone zą ku sobie i połączone, tak u wierzchu, jako i w pośrodku swej wysokości, krzyżami ukośnymi i belkami poziomymi, które zarazem mają złużyć za rusztowania do umieszczenia na nich potrzebnych kołowrotów czyli wind, i składania ciężarów.
56. Każda noga, której dolną część, fig. 17 w widoku i przecięciu, figura zaś 18 w planie wiobraża, ma kształt ostrokręgu ściętego, nieco pochyło wygiętego, około 30 st. (8,6 m.) średnicy w podstawie trzymającego, oraz wydrążonego wewnątrz; i złożona jest z belek albo raczej z półczyzn czyli bali 6calowych (0,14 m.), (większa bowiem grubość nie jest potrzebna), stawianych jednych na drugich i z sobą stykających się, a przytem tak składanych, aby jedne pomiędzy drugie wchodziły i tym sposobem wiązały się

z sobą. W tym celu najprzód wbite mają być pale ABC (fig. 13) 12 cali (0,29 m.) w kw. grube, stanowiące podstawę nogi, z których po dwa na przemian jedne w wysokości D, drugie w E, poucinane zostaną. Na nich stawiane będą bale DF, EG..., tak aby podobnież na przemian jedne wyżej nad drugie wystawały, co w całej długości nogi ma się powtarzać. Ażeby te wszystkie części stanowiły całość, nie podlegającą zmianie kształtu, utrzymane zostaną wewnątrz za pomocą kół, w miejscach DD, EE, FF, GG,... co 20 st. (5,8 m.) jedno od drugiego przypadających, do których końce bali mocnymi gwoździami przybite będą. W składaniu kół postawi się naprzód złup środkowy SS, dla utrzymania belek krzyżowych HI, KL, z którymi łączą się ramiona albo promienie ZM, ZM,.. Wszystkie promienie przedłużone zą na kilka stóp dla zrobienia na nich rusztowań zewnętrznych, wewnątrz zaś przy obwodzie połączone wieńcem NOP, zbitym z bali, podobnie jak w kołach wodnych lub zwyczajnych buksztelach. Za każdem postawieniem w ścianie jednego rzędu bali, powinno być złożone takie koło w końcach tychże n. p. G, G..., dla zrobienia rusztowania RR, z któregoby nowy rzęd bali pomiędzy pierwszymi ustawić można było. {Wszystkie te bale będą jeszcze, wpośrodku swej długości pomiędzy dwoma kołami, łączone z sobą żelaznymi klamrami k, k; przestwory zaś między tymiż balami, gdzie potrzeba, wypełniane listwami lub klinami.

Potrzebne pochylenie nogi kozła można łatwo otrzymać przez wbijanie, gdzie należy, klinów pomiędzy stykające się końce bali. - Cała zresztą robota jest nader prosta i nie trudniejsza od innych w zwyczajnych konstrukcjach, jak tylko przez to, że na większą wysokość ciężary podnoszone być muszą, co, jak niżej się okaże, przy użyciu wind jest mało znaczącą rzeczą.

Moc podpór czyli kozłów.

57. Podług doświadczeń Rondeleta do zgniecenia pręta krótkiego sosnowego 1 cal (0,024 m.) grubości w kwadrat mającego, potrzeba około 70 cetn. (2 800 kil.) (*). {(*){{Art de âbtir.- Navier: Application de la Mecanique etc.}}} W konstrukcjach jednak, takich nawet w których drzewo najbardziej bywa obciążane (jak n.p. w palach fundamentowych), rzadko więcej liczy się na moc jednego cala kw. jak 6 cet. (0,4 kil. na 1 mm. kw.). - Kiedy długość pręta drewnianego lub belki, około 20 do 24 razy większa jest od jego grubości, wówczas, podług doświadczeń tegoż Rondeleta, moc drzewa pionowo obciążonego prawie dwa razy się zmniejsza. {W nogach kozłów opisanych takiż jest właśnie stosunek długości (10 st. = 2,88 m.), na jaką bale, między kołami i klamrami, bez utrzymania zostają, do grubości tychże bali (6 c.=0,144 m.). Moc zatem onych wyniesie 3 cet. na 1 cal kw., albo około 400 cet. na 1 st. kw. (200 000 kil. na 1 m. kw.). Gdy powierzchnia przecięcia bali, składających niższą część nogi, najmniej 45 st. kw. (3,6 m. kw.) czyni, na całą przeto moc jednej nogi albo ciężar jaki takowa może dźwigać, wypada 18000 cet. (720 000 k.), czterech zaś nóg albo jednego kozła (72 000 cet. (2 880 000 kil.)

58. Największy ciężar części arkady spoczywającej na jednym koźle, wynosić może wraz z rusztowaniami wiszącymi . . . . 30 000 cet. (1 200 000 kil.). Ciężar kozła i rusztowań do niego należących . . 40 000 cet. (1 600 000 kil.).

Ogół ciężaru w podstawie kozła, wynosi . . . . 70 000 cet. (2 800 000 kil.); co prawie wyrównywa powyższej mocy (*) {(*) U wierzchu kozła, ciśnienie zmniejszone jest przynajmniej o połowę, o tyle też zatem zmniejszy się i ilość drzewa, oraz średnica nóg, która w tem miejscu wynosi 15 stóp.}.

ROZDZIAŁ IKs.

O TRACHYTACH I BASALTACH.

Dawne te Lawy wulkanicznych roztopionych skał, stanowią osobną i obszerną dzielnicę niepodległą ({{terrains independans}}) skał zagadliwych, ({{problematiques}}), jakimi zą Trachyty i Basalty ; massy lite z feldspatu i Hornblendy, (rogomienia) albo z feldspatu i Augitu (1) {(1) Augit, ({{èPyroksne}}) nazwany tak od blasku; więc spolszczyć go można na blasczeń.} złożone, które Cordier nazwał skałami ogniorodnymi ({{roches èpyrognes}}), a które przez niego i przez Fleriau de Bellevue oryktognostycznie rozkładane (2) {(2) {{Journal de Physique}} Tom LI. LKs. i LKsKsIII.}, dotąd w sposobie powstania swoiego nie zą docieczone. Liczą się one do utworów wulkanicznych, bo neptunicznych pochód cale wyłączaią, ale żadnego niemaią podobieństwa do Law teraźniejszymi gorzelcami wymiatanych.

Trachyty zą porfyrowatymi skałami częstokroć dziurkowatymi, chropowatymi na dotknięcie i przeto tak przez Hauj nazwane, Trachys, szorstki, nierówny. (1) {(1) Można go nazwać szorścień.}

Dolomieu policzył ie do granitowych i porfyrowatych Law, a Buch nazwał ie Trappo Porfyrami. Zą one albo same albo złączone z innymi oryktognostycznie różnymi skałami, jakoto Perlity, Obsydiany, gębczenie (2) {(2) Gębczeń, pumeks, pumite, Bimstein nie jest żadnym osobnym rodzajem skały, ale belkowatym, włóknistym stanem w który pewne wulkaniczne skały przechodzą; Hauj odróżnił i w Klassyfikaci swoiej położył Trachyt,

Perlit i Obsydian, jako sposobne do tego stanu przechodzić.} i inne dotąd nieopisane. Nigdzie ich niema w prądach jak nowe Lawy, ale po wszystkich częściach zimi wznoszą się to w grona górzyn, jak Wygorzelec w Węgrzech (Vihoret) od 15 do 20 mil wzdłuż; to W całe pasma lub obszerne płyty ciągłej i potężnej massy litej, jak w płycie Quito od 14 do 18 tysięcy

stóp grubości karni. (1) {(1) Karnią górnicy nasi nazywaią grubość ławic, czyli złog rudy.} albo ogromnych rumowisk i zlepów częstokroć nad bliższe górzyny wznioślejszych, jak W Węgrzech grzbiet między Neusohl i Kremnitz na 3186, i całe pasmo Matra na 2800 stóp wysokie.

Basalty, {{Basanites}}, starożytnych, zą rodzaiem skał czarnych mianych za krzemień mniej wiecej z Rogomieniem (hornblendą) lub z Augitem zmieszany, i uważanych za zbitą rozmaitość Zielonka, (ügrnstein) który cześcią w Diabasy i w Trappicie Brogniarta, częścią w Diorycie i Afanicie Haujego zawiera się. Dla tego że skały te łatwo z innymi mieszaią się i trudne zą do rozeznania, Brogniart nazwał ie {{mimose}} od {{mimos}} naśladowanie, o Hauaj Dolerit od {{dorlos}} obłuda; niepodobna zaś częstokroć jest rozpoznać, czyli ta massa czarna jest rogomieniem (Hornblendą) lub Augitem. Co Basalty szczególnie od wszystkich innych skał, a mianowicie od Zielonka i Trachytu w składzie swoim odróżnia, jest Oliwień (olivin) którego w tamtych skałach cale niema; Augit zaś i magnesowe żelazo we wszystkich znachodzone bywa. Niema też nigdy w basaltowych złogach ani gębczenia ani Obsydianu, ani właściwego Trachytu; owszem obiedwie te skały zdaią się na wzaiem odpychać, nigdzie się niełączą, wszędy osobno stoią, a gdzie zą sobie najbliżej, tam Basalt na Trachycie leży lub od niego zlepami gębczenia albo Trachytu odłączony. Często zaś Basalt w pierwotnych skałach widziany starszym być może od niektórych Trachytów. Nakoniec, to jeszcze Basalty różni od Trachytów, iż się w prądach jak terazniejsze Lawy znajduią; tak mianowicie w Montpezat, gdzie zapełniaiąc dolinę padły, jak się wydaie na kałuże i nagłem parowaniem wody utworzyły sklepiste podniesienia miejscami w złupy połupane; za tym widocznym prądem dochodzi się osobno stoiącego wzgórza, którego spusty kulastymi zuzlami zą okryte, a na wierzchołku widać ostatki doskonałego Krateru. (1) {(1) {{Faujas de St Fond: Sur les Volcans eteints du Vivarais.}}} Lubo zaś nie wszędy tak

wyraźne zą basaltowe prądy, zawdy jednak te same na sobie noszą znamiona.

Długo czarne i zielone głazy pomników i obelisków egipskich uchodziły, zwłaszcza u Antykwariuszów, za Basalty. Mineralogowie, i Werner sam, uwiedziony złomkami, miał ie za Sjenit; nowsze śledzenia P. Wad na najpiękniejszym w tym względzie obelisku w Rzymie na Piazza Navona znajduiącym się, i P. èRozires w okolicach Sjeny, na półwyspie góry Synai i w kamienistej Arabi dowiodły: że skały te, z których podług wszelkiego podobieństwa Tablice prawa Mojżeszowe zrobione były i jak wieść niesie w teraźniejszym Dziebel-Musa maią być zakopane, zą przechodnim tworem granitycznym z rogomienia (Hornblendy) i krzemienia, z nieco kwarcu i błyszczu wmieszanego złożonym i dokładnie Synajtem nazwać się mogącym.

Wulkaniczne te utwory rozmaity wiek powstania swego wyraznie okazuią, i wszędy do przechodnich skał należeć się zdaią; to ie zagadliwymi czyni i nasuwa wniosek powolnego przeięcia od dołu do góry, oraz ścisły związek między najnowszymi złogami przechodnimi, najdawniejszym osadem i dawnymi wulkanicznymi tworami. Gdy zaś Trachyt sam rzadko wolno na świat wychodzi i częstokroć pokryty jest piasczeniem i nowymi wapieniami, które znowu pokryte zą wapieniem jeziernym z małzami; stąd pozniejszy jego wiek i porę powstania między tworem osadu i tak zwanymi trzeciakimi gruntami wniesiono.

A.0.3. Reguły

```
# # Auxilliary grammar file
# ## All bytes
```

```
export Any = Optimize[ "[1]" | "[2]" | "[3]" |  "[4]" |  "[5]" |  "[6]" |
    "[7]" |  "[8]" |  "[9]" | "[10]" | "[11]" | "[12]" | "[13]" | "[14]" |
    "[15]" | "[16]" | "[17]" | "[18]" | "[19]" | "[20]" | "[21]" | "[22]" |
    "[23]" | "[24]" | "[25]" | "[26]" | "[27]" | "[28]" | "[29]" | "[30]" |
    "[31]" | "[32]" | "[33]" | "[34]" | "[35]" | "[36]" | "[37]" | "[38]" |
    "[39]" | "[40]" | "[41]" | "[42]" | "[43]" | "[44]" | "[45]" | "[46]" |
    "[47]" | "[48]" | "[49]" | "[50]" | "[51]" | "[52]" | "[53]" | "[54]" |
    "[55]" | "[56]" | "[57]" | "[58]" | "[59]" | "[60]" | "[61]" | "[62]" |
    "[63]" | "[64]" | "[65]" | "[66]" | "[67]" | "[68]" | "[69]" | "[70]" |
    "[71]" | "[72]" | "[73]" | "[74]" | "[75]" | "[76]" | "[77]" | "[78]" |
    "[79]" | "[80]" | "[81]" | "[82]" | "[83]" | "[84]" | "[85]" | "[86]" |
    "[87]" | "[88]" | "[89]" | "[90]" | "[91]" | "[92]" | "[93]" | "[94]" |
    "[95]" | "[96]" | "[97]" | "[98]" | "[99]" | "[100]" | "[101]" | "[102]" |
    "[103]" | "[104]" | "[105]" | "[106]" | "[107]" | "[108]" | "[109]" |
    "[110]" | "[111]" | "[112]" | "[113]" | "[114]" | "[115]" | "[116]" |
    "[117]" | "[118]" | "[119]" | "[120]" | "[121]" | "[122]" | "[123]" |
    "[124]" | "[125]" | "[126]" | "[127]" | "[128]" | "[129]" | "[130]" |
    "[131]" | "[132]" | "[133]" | "[134]" | "[135]" | "[136]" | "[137]" |
    "[138]" | "[139]" | "[140]" | "[141]" | "[142]" | "[143]" | "[144]" |
    "[145]" | "[146]" | "[147]" | "[148]" | "[149]" | "[150]" | "[151]" |
    "[152]" | "[153]" | "[154]" | "[155]" | "[156]" | "[157]" | "[158]" |
    "[159]" | "[160]" | "[161]" | "[162]" | "[163]" | "[164]" | "[165]" |
    "[166]" | "[167]" | "[168]" | "[169]" | "[170]" | "[171]" | "[172]" |
    "[173]" | "[174]" | "[175]" | "[176]" | "[177]" | "[178]" | "[179]" |
    "[180]" | "[181]" | "[182]" | "[183]" | "[184]" | "[185]" | "[186]" |
    "[187]" | "[188]" | "[189]" | "[190]" | "[191]" | "[192]" | "[193]" |
    "[194]" | "[195]" | "[196]" | "[197]" | "[198]" | "[199]" | "[200]" |
    "[201]" | "[202]" | "[203]" | "[204]" | "[205]" | "[206]" | "[207]" |
    "[208]" | "[209]" | "[210]" | "[211]" | "[212]" | "[213]" | "[214]" |
```

```
    "[215]" | "[216]" | "[217]" | "[218]" | "[219]" | "[220]" | "[221]" |
    "[222]" | "[223]" | "[224]" | "[225]" | "[226]" | "[227]" | "[228]" |
    "[229]" | "[230]" | "[231]" | "[232]" | "[233]" | "[234]" | "[235]" |
    "[236]" | "[237]" | "[238]" | "[239]" | "[240]" | "[241]" | "[242]" |
    "[243]" | "[244]" | "[245]" | "[246]" | "[247]" | "[248]" | "[249]" |
    "[250]" |"[251]" | "[252]" | "[253]" | "[254]" | "[255]"];

# ## Character Rules
# _Character Rules_ are responsible for listing all characters that may appear
    in Polish texts:
# historical and temporary

Lower = "a" | "ą" | "á" | "b" | "c" | "ć" | "d" | "e" | "é" | "ę" | "f" | "g"
    | "h" | "i" | "j" | "k" | "l" | "ł" | "m" | "n" | "ń" | "o" | "ó" | "p" |
    "q" | "r" | "s" | "ś" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | "ź" |
    "ż" ;
Upper = "A" | "Ą" | "Á" | "B" | "C" | "Ć" |"D" | "E" | "É" |"Ę" |"F" | "G" |
    "H" | "I" | "J" | "K" |"L" | "Ł" |" M" | "N" | "Ń" | "O" |"Ó" | "P" | "Q"
    | "R" | "S"| "Ś" | "T" | "U" | "V" |"W" | "X" | "Y" | "Z"| "Ź" | "Ż" ;
Letter = Lower | Upper ;
LowerVowel = "a" | "ą" | "á" | "e" | "é" | "ę" | "i" | "o" | "ó" | "u" | "y" ;
UpperVowel = "A" | "Ą" | "Á" | "E" | "É" | "Ę" | "I" | "O" | "Ó" | "U" | "Y" ;
Vowel = LowerVowel | UpperVowel ;
LowerConsonant = Lower - LowerVowel ;
UpperConsonant = Upper - UpperVowel ;
Consonant = LowerConsonant | UpperConsonant ;
AnyString = Letter* ;
NonEmptyString = Letter+ ;
NonLetter = Any - Letter ;
```

```
Beginning = "[BOS]" | NonLetter ;
End = "[EOS]" | NonLetter ;

# ## Palatalization issues
# The below automaton list consonants that are followed by the "j" character,
    if palatelized.
Followed_by_j = "c" | "s" | Vowel "z" | "C" | "S" | Vowel "Z" ;

# The below automaton list characters that are followed by the "i" character,
    if palatelized.
Followed_by_i = "d" | "f" | "g" | "h" | "k" | "l" | "m" | "n" | "p" | "r" |
    "t" | "w" | "D" | "F" | "G" | "H" | "K" | "L" | "M" | "N" | "P" | "R" |
    "T" | "W" ;

# ## Capitalization issue
# The rules of the type "Ll" deal with capitalization issue (lower, upper
    character) for standard letters.
# The rules of the type "Caca" deal with capitalization issue for acute
    letters (e. g. "ć").
# The rules of the type "Anan" deal with capitalization issue for nasal vowels
    (e. g. "ą").
# The rules of the type "Lclc" deal with capitalization issue for other Polish
    diacritic letters (e. g. "ż").

Aa = ( "A" | "a" ) ;
Anan = ( "Ą" | "ą" ) ;
Aaaa = ( "Á" | "á" ) ;
Bb = ( "B" | "b" ) ;
Cc = ( "C" | "c" ) ;
```

```
Caca = ( "Ć" | "ć" ) ;
Dd = ( "D" | "d" ) ;
Ee = ( "E" | "e" ) ;
Enen = ( "Ę" | "ę" ) ;
Ff = ( "F" | "f" ) ;
Gg = ( "G" | "g" ) ;
Hh = ( "H" | "h" ) ;
Ii = ( "I" | "i" ) ;
Jj = ( "J" | "j" ) ;
Kk = ( "K" | "k" ) ;
Ll = ( "L" | "l" ) ;
Lclc = ( "Ł" | "ł" ) ;
Mm = ( "M" | "m" ) ;
Nn = ( "N" | "n" ) ;
Nana = ( "Ń" | "ń" ) ;
Oo = ( "O" | "o" ) ;
Oaoa = ( "Ó" | "ó" ) ;
Pp = ( "P" | "p" ) ;
Qq = ( "Q" | "q" ) ;
Rr = ( "R" | "r" ) ;
Ss = ( "S" | "s" ) ;
Sasa = ( "Ś" | "ś" ) ;
Tt = ( "T" | "t" ) ;
Uu = ( "U" | "u" ) ;
Vv = ( "V" | "v" ) ;
Ww = ( "W" | "w" ) ;
Xx = ( "X" | "x" ) ;
Zz = ( "Z" | "z" ) ;
Zaza = ( "Ź" | "ź" ) ;
```

```
Zczc = ( "Ż" | "ż" ) ;
# # Feliński's Rules (before 1830)

# ## Auxiliary rules (imported from 000-md.grm)
# ### Letter = LoadFstFromFar['grammars/000-far', 'Letter'];
# ### AnyString = LoadFstFromFar['grammars/000-far', 'AnyString'];
# ### NonEmptyString = LoadFstFromFar['grammars/000-far', 'NonEmptyString'];
# #    import 'grammars/000-grm' as aux ;

# ## Rule: A_acute (the letter no longer exists in Polish)
# This rule replaces 'a_acute' with standard 'a'.
A_acute = (("Á" : "A") | ("á" : "a")) ;

export Rule001_01 = Optimize[ CDRewrite[ A_acute, AnyString, AnyString, Any* ]
    ];

# ## Rule: Infinitive 'ź_into_ć'
# This rule replaces 'ź' that ends verb infinitives with 'ć'.
# The words "wiedź" and "kładź" have been removed from tests, as they exist in
    modern vocabulary
# > bydź -> być
# > uledz -> ulec
# > piedz -> piec
# > módz -> móc
Infinitive_z ="y" ("dź" : "ć") | ("ie" | "a") ("dź" : "ść") | ("e" | "ó")
    ("dz" : "c") ;

export Rule001_02 = Optimize[ CDRewrite[ Infinitive_z, NonEmptyString, End,
    Any* ] ];
```

```
# ## Rule: Imperative 'iej'
# This rule replaces imperatives ending "iej" into "ij"; "rzej"" into "rzyj".
# The word "zwiej" has been removed from tests, as it exists in modern
    vocabulary
# > rwiej -> rwij
# > tniej -> tnij
# > drzej -> drzyj
Imperative_iej =("tniej" : "tnij")| ("rwiej" : "rwij")| ("drzej" : "drzyj") ;

export Rule001_03 = Optimize[ CDRewrite[ Imperative_iej, AnyString, End, Any*
    ] ];

# ## Rule: Preposition 'z'
# These rules replace 's' at the end of a preposition with 'z'.
# The below rule handles bare prepositions.
# > bes -> bez
# > s -> z
# > przes -> przez
Preposition_z =Bb ("es" : "ez")| Pp ("rzes" : "rzez" )| ( ("s" : "z") | ("S" :
    "Z") ) ;

export Rule001_04 = Optimize[ CDRewrite[ Preposition_z, Beginning, End, Any* ]
    ];

# The below rule handles bare prepositions that prefix nouns.
# > beskresie -> bezkresie
# > roskoszy -> rozkoszy
Preposition_z_inflect = Bb("eskres" : "ezkres") | Rr("oskosz" : "ozkosz") ;
```

```
export Rule001_05 = Optimize[ CDRewrite[ Preposition_z_inflect, AnyString,
    AnyString, Any* ] ];

# ## Rule: Letter 'j' before 'i'
# When 'y' is followed by 'i' then: 'y' disappears.
# The rule also processes the word "szyyi", which is removed from tests, as it
    is finally and uncorrectly converted into "szji" according to rule 005_01.
# > krayina -> kraina
# > stoyisz -> stoisz
Jota_before_i = ("y": "") "i" ;

export Rule001_06 = Optimize[ CDRewrite[ Jota_before_i, NonEmptyString,
    AnyString, Any* ] ];

# ## Rule: Letter 'j' before vowel other than 'i'
# The rule replaces 'y', occurring before vowel other than 'i', with 'j'.
# > Yabłko -> Jabłko
# > swoye -> swoje
Jota_before_vowel =( ("y" : "j") | ("Y" : "J") )(Vowel - "i");

export Rule001_07 = Optimize[ CDRewrite[ Jota_before_vowel, AnyString,
    AnyString, Any* ] ];

# Otherwise the rule replaces 'y' occurring after vowel with 'j'.
# > bayka -> bajka
# > kray -> kraj
Jota_after_vowel =Vowel("y" : "j") ;
```

```
export Rule001_08 = Optimize[ CDRewrite[ Jota_after_vowel, AnyString,
    AnyString, Any* ] ];

# ## Rule: Letter 'i' before vowel
# The rule replaces 'i', occurring at the beginning of the word and before
    vowel, with 'j'.
# > iayko -> jajko
Jota_at_beginning =( ("i" : "j") | ("I" : "J") ) Vowel ;

export Rule001_09 = Optimize[ CDRewrite[ Jota_at_beginning, Beginning,
    NonEmptyString, Any* ] ];



# # Delagacyja Rules (1830)
# These are rules introduced by Akademia Umiejętności in 1830

# ## Rule: O_acute
# The rule replaces 'ó' (o_acute) with 'o' in selected words
# > aktór -> aktor
# > honór > honor
# > fawór -> fawor
# > dzięciół -> dzięcioł
# > gruczół -> gruczoł
# > mozóły -> mozoły
O_acute = ( Aa ( "któr" : "ktor" ) | Hh ( "onór" : "onor" ) | Ff ( "awór" :
    "awor" ) | Dd ( "zięció" : "zięcio" ) ( "ł" | "l" ) | Gg ( "ruczó":
    "ruczo" ) ( "ł" | "l" ) | Mm ( "ozó" : "ozo" ) ( "ł" | "l" ) Vowel) ;
```

```
export Rule002_01 = Optimize[ CDRewrite[ O_acute, Beginning, AnyString, Any* ]
    ];

# The word ód has been removed from tests, as it exists in modern vocabulary
# > pód -> pod
O_acute_end = (( "ód" : "od" ) |( "Ód" : "Od" ) | Pp ( "ód" : "od" ) ) ;

export Rule002_02 = Optimize[ CDRewrite[ O_acute_end, Beginning, End, Any* ] ];

# ## Rule: Polonization of foreign names
# The rule reverts the original spelling of foreign names.
# The rule changes Marija int Marya, which then finally is converted into Maria
# > Marija -> Maria
# > Maryja -> Maria
# > Julija -> Julia
# > Fabijan -> Fabian
# > Scypijon -> Scypion
# > Grecyja -> Grecya
Foreign_names = ( ( "Marij" : "Mary" ) | ( "Maryj" : "Mary" ) |( "Julij" :
    "Juli" ) | ( "Fabijan" : "Fabian" ) |( "Scypijon" : "Scypion" ) | (
    "Niobe" : "Nyobe" ) |("Grecyj" : "Grecy")) ;

export Rule002_03 = Optimize[ CDRewrite[ Foreign_names, Beginning, AnyString,
    Any* ] ];

# ## Rule: Foreign words with "gie"
# The rule deletes 'i' from "gie" in the foreign words.
# > gienerał -> generał
# > geniusz -> geniusz
```

```
# > Eugieniusz -> Eugeniusz
# > Gienewy -> Genewy
Foreign_names_with_ge = (( "Eugieniusz" : "Eugeniusz" ) | Ee ("wangieli" :
    "wangeli" ) | Gg ("ie" : "e") "nerał" | Gg ("ie" : "e") "niusz"| Gg ("ie"
    : "e") "ografi" | Gg ("ie" : "e") "nealogi"| Gg ("ie" : "e") "ometr" |
    ("Gie" : "Ge") "rmani"| ("Gie" : "Ge") "new"| ("Gie" : "Ge") "rtrud" ) ;

export Rule002_04 = Optimize[ CDRewrite[ Foreign_names_with_ge, Beginning,
    AnyString, Any* ] ];

# ## Rule: Past Participle "łszy"
# The rule inserts 'ł' before "szy" in past participle forms
# > poszedszy -> poszedłszy
# > usiadszy -> usiadłszy
# > zjadszy -> zjadłszy
# > zdarszy -> zdarłszy
# > zaniósszy -> zaniósłszy
# > znalazszy -> znalazłszy
Past_participle = (( "dszy" : "dłszy" ) |( "rszy" : "rłszy" ) | ( "sszy" :
    "słszy" ) |( "zszy" : "złszy")) ;

export Rule002_05 = Optimize[CDRewrite[ Past_participle, NonEmptyString, End,
    Any* ] ];

# ## Rule: Remove 'z' from 'źrz"
# The rule removes 'z' form "źrzódło" etc.
# > źrzódło-> źródło
# > śrzoda -> środa
```

```
Remove_z_from_zrz = (( "śrz" : "śr" ) |( "Śrz" : "Śr" ) | ( "źrz" : "źr" ) |(
    "Źrz" : "Źr" )) ;

export Rule002_06 = Optimize[CDRewrite[ Remove_z_from_zrz, AnyString,
    AnyString, Any* ] ];

# ## Rule: "adjective_endings"
# The rule replaces obsolete adjective endings: "iemi" with "imi"; "emi" with
    "ymi"
# > wysokiemi -> wysokimi
# > długiemi -> długimi
# > nowemi -> nowymi
# > dobremi -> dobrymi
Adjective_endings = (("iemi" : "imi") |("emi" : "ymi")) ;

export Rule002_07 = Optimize[CDRewrite[ Adjective_endings, NonEmptyString,
    End, Any* ] ];

# # Akademia Rules (1891)
# These are rules introduced by Akademia Umiejętności in 1891

# ## Rule: E_acute (the letter no longer exists in Polish)
# This rule replaces 'e_acute' with standard 'e'.
E_acute = (("É" : "E") |("é" : "e")) ;

export Rule003_01 = Optimize[CDRewrite[ E_acute,AnyString,AnyString,Any* ] ];

# # Rules by Akademia Umiejętności in Cracow, 1918
# ## Rule: palatalization of preposistion 'z' in verbs
```

```
# The rule converts 'z' into 'ś' in verbs starting with 'ci'
# > zcierpieć -> ścierpieć
# > zcisnąć -> ścisnąć
Soft_z =( ("z" : "ś") | ("Z" : "Ś") )"ci" ;

export Rule004_01 = Optimize[CDRewrite[ Soft_z, Beginning, NonEmptyString,
    Any* ] ];
# # Rules form 1936

# ## Rule: Ending_ja
# The rule replaces endings: "yja" with "ja", "yj" with "ji".
# The words 'decyzyj', 'lekcyj', 'dygresyj' have been removed from tests, as
    they exist in modern vocabulary
# > decyzyja -> decyzja
# > decyzyje -> decyzje
# > lekcyja -> lekcja
# > degresyja -> degresja
Ending_yj = ( ( "yj" | "yi") : "ji" ) ;

export Rule005_01 = Optimize[ CDRewrite[ Ending_yj, Followed_by_j, End, Any* ]
    ];

Ending_yja = ( ("yj" | "yi") : "j" ) ("a" | "i" | "ę" | "ą" | "e" | "om" |
    "ami" | "ach" ) ;

export Rule005_02 = Optimize[ CDRewrite[ Ending_yja, Followed_by_j, End, Any*
    ] ];

# ## Rule: Ending_iake
```

```
# The rule replaces ending "yja" with "ia", "yj" with "ii"
# The words 'historyj', 'religij', 'geografij' have been removed from tests,
    as they exist
# > historyja -> historia
# > historyje -> historie
# > kopyj -> kopii
# > religija -> religia
Ending_ij = ( ( "yj" | "ij" ) : "ii" ) ;

export Rule005_03 = Optimize[ CDRewrite[ Ending_ij, Followed_by_i, End, Any* ]
    ];

Ending_ija =( ( "yj" | "ij" ) : "i" )("a" | "i" | "ę" | "ą" | "e" | "om" |
    "ami" | "ach" ) ;

export Rule005_04 = Optimize[ CDRewrite[ Ending_ija, Followed_by_i, End, Any*
    ] ];

# ## After 1936 it was agreed that after all consonants (except for "c', "s",
    "z"), "y" or "j" are replaced by "i"
# > Danya -> Dania
# > Danja -> Dania
I_after_consonant = ( ("y" | "j") : "i" ) ;

export Rule005_05 = Optimize[ CDRewrite[ I_after_consonant, Followed_by_i,
    Vowel, Any* ] ];

# ## The rule 005_06 has been removed to 900-One2N.md.
# ## The rules 005-07 - 005-13 have been removed to 901-N2One.md.
```

```
# ## Rule 'Xi'
# The rule replaces "xi" with "ksi".
Replace_xi =( ("X":"K") | ( "x" : "k" ) ) ("i" : "si") ;

export Rule100_01 = CDRewrite[ Replace_xi, AnyString, ("ą" | "ę"), Any* ] ;

# ## Rule 'Puhar'
# The rule replaces "Puhar" with "Puchar"
Replace_puhar =Pp("uhar": "uchar") ;

export Rule100_02 = Optimize[ CDRewrite[ Replace_puhar, Beginning, AnyString,
    Any* ] ];

# The file gathers rules that divide words. Names of the rules point out from
    their origin, e.g., the prefix 005 denotes that the rule belongs to the
    005 family (rules changed in 1936).

# ## Adjective masculine endings in instrumental and locative case have been
    unified:
# ## "ymi", "emi" -> "ymi" ; This rule has been handled by Felinski
# ## "ym", "em" -> "ym" ; This rule is doubtful as there are too many
    exceptions
# ## The problem appeared with prepositions: "potem", "zaczem":
# ## The below rule handles prepositions ending with "czem".
# The word 'zaczem' has been removed from tests, as it exists in modern
    vocabulary
# > poczem -> po czym
# > przyczem -> przy czym
```

```
Preposition_before_czem = Zz("aczem" : "a czym") | Pp("oczem" : "o czym") |
    Pp("rzyczem" : "rzy czym") ;

export Rule005_06 = Optimize[ CDRewrite[ Preposition_before_czem, Beginning,
    End, Any* ] ];

# ## Particle "by"
# "By" particles and their inflected forms should remain merged with verbs,
    and separated otherwise (there are exceptions!)

# ### Inflected singular forms of "by"
Singular_by = ("bym" | "byś" | "by") ;

# ### Inflected plural forms of "by"
Plural_by = ("byśmy" | "byście" | "by") ;

# ### All inflected forms of "by"
Ending_by = ( Singular_by | Plural_by );

# ### Processing
# First, separate all 'by' endings from "roots". The separation is done via
    the auxilliary symbol '&'.
SeparateSingular_by = ("": "&") Singular_by ;
SeparatePlural_by = ("": "&") Plural_by ;
Separate_by = ( SeparateSingular_by | SeparatePlural_by ) ;

export Rule005_07 = Optimize[ CDRewrite[ Separate_by, NonEmptyString, End,
    Any* ] ];
```

```
# Merge all verb roots with corresponding endings:
#  if the roots are past forms of words:
#   merge roots with "by" endings
#  else: divide words

# #### Singular forms
PastSingularEnding = ( "ł" | "ła" );
ConcatSingular = PastSingularEnding ("&": "") Singular_by ;

export Rule005_08 = Optimize[ CDRewrite[ ConcatSingular, AnyString, End, Any*
    ] ];
# #### Plural Forms
PastPluralEnding = ("li" | "ły" ) ;
ConcatPlural = PastPluralEnding ("&": "") Plural_by ;

export Rule005_09 = Optimize[ CDRewrite[ ConcatPlural, AnyString, End, Any* ]
    ];

# ### Conjuctions that are merged with the particle "by".
# Besides past forms of verbs, also some conjuctions should by merged with the
    particle 'by'.
Conjunction_by = Beginning ( Aa | Aa"le" | Aa"że" | Zczc"e" | Ii"ż" | Gg"dy" |
    Cc"hociaż" | Jj"ak" | Cc"zy" | Cc"zyż" | Nn"iechaj" | Nn"i" | Oo |
    Jj"akkolwiek" | Nn"iechaj" | Nn"im" ) ;
ConcatConjunction = Conjunction_by ("&": "") Ending_by ;

export Rule005_10 = Optimize[ CDRewrite[ ConcatConjunction, AnyString,End,
    Any* ] ];
```

```
# All words that end with 'by" (and alike), and are not conditional words or
    conjuctions listed above, should be divided into two words.
Separate = AnyString ("&": " ") AnyString ;

export Rule005_11 = Optimize[ CDRewrite[ Separate, AnyString, End, Any* ] ];

# ## "Nie + modal verbs
# The particle "nie" should be separated from modal verbs.
SeparateModal = Nn"ie"("" : " ")("można" | "trzeba" | "wolno" | "powinno" |
    "potrzeba" ) ;

export Rule005_12 = Optimize[ CDRewrite[ SeparateModal, Beginning, End, Any* ]
    ];

# ## Specific rules for word division
# These are rules that divide words into parts (usually: preposition + noun)
SeparateWord =( "bez"(": " ") ("mała" | "wątpienia") | "beze"(": " ") "mnie"
    | "do"(": " ") ("cna" | "syta" | "niedawna") | "na"(": " ") ("pewno" |
    "czczo" | "jaw" | "ogół" | "podorędziu" | "przemian" | "zabój" ) | "nade"
    (": " ") "wszystko" | "od" (": " ") ("niechcenia" | "razu" ) | "ode"(":
    " ") "mnie" | "po" (": " ") ("cichu" | "ciemku" | "kryjomu" | "społu" |
    "trochu" | "troszku") | "pode" (": " ") "mną" | "przede" (": " )
    ("dniem" | "wszystkim") | "w" (": " ") "ogóle" | "we" (": " ") "mnie" |
    "z" (": " ") ("powodu" | "pomocą" | "cicha" | "głupia" | "daleka" |
    "dala" ) | "za" (": " ") ("mąż" | "pomocą" | "świeża" | "widna" ) | "ze"
    (": " ") "mną") ;

export Rule005_13 = Optimize[ CDRewrite[ SeparateWord, Beginning, End, Any* ]
    ];
```