Transfer w systemie POLENG-3

Transfer in the POLENG-3 system

Krzysztof Jassem

Uniwersytet im. Adama Mickiewicza
Wydział Matematyki i Informatyki
e-mail: jassem@amu.edu.pl

STRESZCZENIE

Praca przedstawia wybrane etapy procesu tłumaczenia wyrażenia języka polskiego na język angielski w systemie POLENG-3. Są to: selekcja jednej spośród hipotetycznych reprezentacji wyrażenia polskiego otrzymanych w wyniku analizy syntaktycznej, transfer wybranej reprezentacji na strukturę wyrażenia angielskiego, modyfikacja składniowa struktury wyrażenia angielskiego i synteza morfologiczna wyrażenia angielskiego. Dwa pierwsze etapy procesu tłumaczenia, tj. analiza leksykalna i analiza składniowa przedstawione są odpowiednio w pracach [1] i [2].

ABSTRACT

The paper presents selected phases of translation process in the POLENG system. The phases are: selection of the appropriate representation from among the ones obtained as a result of a parse of the input Polish expression, transfer of the representation, syntax modification of the transferred representation, and morphological synthesis. The first two steps of the translation process, i.e. lexical analysis and syntax analysis are described in [1] and [2] respectively.

1. Modularyzacja

W przeciwieństwie do poprzednich wersji systemu POLENG (por. [3], [4]) w algorytmie translacji systemu POLENG-3 dokonano podziału zadań na rozdzielne moduły, które przetwarzają dane sekwencyjnie. Poszczególne moduły wykonują odpowiednio następujące zadania: analiza leksykalna i morfologiczna wyrażenia polskiego, analiza składniowa wyrażenia polskiego, wybór najlepszej reprezentacji składniowej wyrażenia polskiego, transfer reprezentacji składniowej wyrażenia polskiego na reprezentację wyrażenia angielskiego, modyfikacja składniowa reprezentacji wyrażenia angielskiego, synteza morfologiczna wyrażenia angielskiego, generowanie powierzchniowe wyrażenia angielskiego. Analiza leksykalna i morfologiczna opisana jest w [1], problemy analizy składniowej przedstawia

[2]. W poniższej pracy opisane zostaną pozostałe moduły (poza generowaniem powierzchniowym).

2. Wybór najlepszej struktury składniowej

W systemie POLENG-3 zakłada się, że w wyniku analizy składniowej otrzymuje się wiele alternatywnych struktur frazowych wyrażenia wejściowego – w tym również takich reprezentacji, które są zbiorem drzew struktury frazowej nieposiadających wspólnego korzenia (taka sytuacja ma miejsce, jeśli w gramatyce systemu nie istnieje symbol, z którego można wyprowadzić analizowane wyrażenie).

W poniższym paragrafie zaproponowano prosty algorytm wyboru jednej struktury składniowej ze zbioru hipotetycznych struktur uzyskanych w wyniku analizy syntaktycznej w oparciu o dwa kryteria:

- kryterium najdłuższego początku,
- kryterium najdłuższej frazy słownikowej.

Pierwsze kryterium stosowane jest wtedy, gdy reprezentacją składniową wyrażenia jest zbiór drzew składniowych reprezentujących fragmenty tekstu – nazywany dalej *lasem*. Wtedy jako lepszy uznaje się ten las, którego najbardziej lewe drzewo obejmuje dłuższy fragment wyrażenia.

Drugie kryterium stosowane jest dla drzew obejmujących ten sam fragment zdania. Jako lepsze uznaje się to drzewo, które zawiera poddrzewo obejmujące dłuższą frazę słownikową (czyli frazę zawartą w słowniku systemu).

Załóżmy, że analizowane zdanie to: *Przynieś tusz do drukarki*. Powierzchniowo możliwe są dwie interpretacje składniowe tego zdania: fraza przyimkowa *do drukarki* może modyfikować czasownik *przynieś* (tego typu interpretacja byłaby logicznie poprawna dla zdania *Przynieś tusz do mojego pokoju*) lub też modyfikować rzeczownik *tusz* (zgodnie ze znaczeniem omawianego zdania). Kryterium najdłuższej frazy pozwala na ustalenie priorytetu drugiej interpretacji, gdyż związane z nią drzewo zawiera poddrzewo obejmujące frazę słownikową długości 3: *tusz do drukarki* (przy założeniu, że fraza ta zawarta jest w słowniku systemu) – podczas gdy drzewo pierwszej interpretacji nie zawiera żadnego poddrzewa obejmującego frazę słownikową.

Algorytm wyboru struktury wyrażenia korzysta z procedury *wybór_drzewa*. Procedura ta definiowana jest tylko dla tych segmentów wyrażenia wejściowego, dla których analiza syntaktyczna zwróciła co najmniej jedną reprezentację w postaci drzewa (a nie lasu) struktury frazowej.

Procedura ‘wybór_drzewa’ – wyboru najlepszego drzewa struktury frazowej dla danego segmentu

Cel: Celem jest wybranie takiego drzewa struktury frazowej, w którym suma długości fraz słownikowych zawartych w poddrzewach jest największa.

Założenie 1: Dla danego segmentu dany jest niepusty zbiór **ZD** drzew struktury frazowej tego segmentu otrzymany w wyniku analizy syntaktycznej.

Założenie 2: W procesie analizy leksykalnej wyznaczono zbiór fraz słownikowych **ZF** – zbudowanych z wyrazów analizowanego wyrażenia (patrz [5]).

Dla danego segmentu zdania procedura zwraca:

- maksymalną sumę długości fraz słownikowych wchodzących w skład poddrzew struktury segmentu – MaxSuma,
- strukturę segmentu (drzewo), dla której suma długości fraz słownikowych zawartych w poddrzewach jest największa - StruktSegmentu

Dane wejściowe

Numer początku i końca segmentu

Dane wyjściowe

MaxSuma, StruktSegmentu

Algorytm procedury

1. MaxSuma := 0
2. **Powtarzaj**
   2.1. Weź kolejną strukturę FP ze zbioru **ZD**
   2.2. *// Porównuj FP z kolejnymi frazami ze zbioru* ***ZF***
   FPJestFrazą := False
   Powtarzaj
   Weź kolejną frazę FL ze zbioru **ZF**
   FPJestFrazą := *zgodność_struktur(FL, FP)*
   //Procedura zgodność_struktur opisana jest w [5]
   Aż (FPJestFrazą) **lub** (Osiągnięto koniec zbioru **ZF**)
   2.3. Jeśli FPJestFrazą
   To
   MaxSuma := Length(S)
   StruktSegmentu := FP
   W pp. *//Szukaj fraz leksykalnych w podfrazach*
   Podziel S na segmenty [S[1],...,S[n]] zgodnie ze strukturą FP.
   For i:=1 to n
   wybór_drzewa(S[i]); zwróć MaxSuma[i]
   SumaPodfraz := Σ(1, n) MaxSuma[i])
   Jeśli SumaPodfraz > MaxSuma
   To
   MaxSuma := SumaPodfraz

StruktSegmentu := FP
Aż (FPJestFrazą) **lub** (Osiągnięto koniec zbioru **ZD**)

Procedura 'wybór_lasu' wyboru najlepszej struktury analizowanego wyrażenia

Procedura ta znajduje zastosowanie dla wyrażeń, które nie zostały w całości rozpoznane w analizie syntaktycznej – ich struktura reprezentowana jest przez las struktur frazowych.

Zastosowana dla wyrażenia, posiadającego strukturę w postaci drzewa, procedura ta sprowadza się do procedury *wybór_drzewa*

Algorytm procedury *wybór_lasu*

wybór_lasu_od_wyrazu (0)

//Procedura wybór_lasu wywołuje procedurę wybór_lasu_od_wyrazu z wartością parametru aktualnego równą '0' i na tym kończy działanie

Algorytm procedury *wybór_lasu_od_wyrazu(n)*

1. Wybierz najdłuższy sparsowany fragment [n, m] zaczynający się w miejscu n
2. wybór_struktury(n, m). Zwróć drzewo StruktSegmentu(n, m)
3. Jeśli m < długość zdania,
   wybór_lasu_od_wyrazu(m). Zwróć Las(m)
4. Połącz drzewo StruktSegmentu i las Las(m). Zwróć las będący połączeniem.

3. Język opisu reguł

Moduł wyboru najlepszej struktury zwraca reprezentację zdania wejściowego w postaci drzewa struktury frazowej. W kolejnych etapach przetwarzania uzyskane drzewo podlega kolejno: transferowi na odpowiadające drzewo struktury angielskiej (na podstawie informacji ze słownika oraz reguł transferu), modyfikacji składniowej (na podstawie reguł rządzących składnią języka angielskiego) oraz syntezie morfologicznej (na podstawie reguł morfologii angielskiej).

Automatyczne wykonanie poszczególnych faz przetwarzania drzewa wymaga opisania wymienionych reguł przy pomocy pewnego formalizmu. Stworzono język opisu reguł, który można zdefiniować przy pomocy gramatyki bezkontekstowej. Język ten jest na tyle ogólny, że można przy jego pomocy opisać wszystkie trzy typy reguł (w rzeczywistości pewne elementy języka wykorzystywane są we wszystkich typach reguł, a niektóre są charakterystyczne tylko dla reguł określonego typu).

Jeśli zdefiniuje się pewien sztuczny język jako oparty na gramatyce bezkontekstowej[1], to łatwe staje się sprawdzenie, czy poszczególne zapisy języka zgodne są z zadaną gramatyką. Można w tym celu posłużyć się

[1] W rzeczywistości BISON wymaga, aby gramatyka spełniała silniejszy warunek – była gramatyką typu LALR(1).

ogólnie dostępnymi narzędziami *flex* i *bison* (patrz [6,7]). Narzędzia te automatycznie generują programy analizujące zgodność wyrażenia języka z zadaną gramatyką.

Dane wejściowe do programu *flex* stanowi plik tekstowy zawierający opis (w odpowiednim formacie) symboli terminalnych dopuszczalnych w gramatyce. Na podstawie tego opisu *flex* generuje kod źródłowy (w języku C) tokenizatora, czyli programu dokonującego podziału dowolnego wyrażenia języka na tokeny (najmniejsze logiczne części języka), podając ponadto charakterystykę semantyczną każdego tokenu (określając na przykład, czy dany token jest wyrazem, liczbą, czy też znakiem interpunkcyjnym).

Dane wejściowe do programu *bison* stanowi plik tekstowy zawierający opis (w odpowiednim formacie) reguł gramatyki. Na podstawie tego opisu *bison* generuje kod źródłowy (w języku C) parsera składniowego, czyli programu dokonującego analizy składniowej (program ten sprawdza poprawność wyrażenia oraz podaje jego strukturę).

Oba programy uzupełniają się następująco: w celu sprawdzenia zgodności wyrażenia z gramatyką w pierwszym etapie wyrażenie poddawane jest działaniu tokenizatora (wygenerowanego przez *flex*), w wyniku czego uzyskuje się reprezentację semantyczną poszczególnych tokenów wyrażenia. Następnie reprezentacja ta staje się wejściem do parsera składniowego wygenerowanego przez *bison*. Parser sprawdza poprawność i w przypadku powodzenia podaje strukturę wyrażenia.

Tak więc, narzędzia *flex* i *bison* umożliwiają sprawdzenie poprawności formalnej omawianych tutaj reguł transferu, modyfikacji i syntezy morfologicznej, czyli ich zgodności z przyjętym formalizmem.

Język reguł systemu POLENG obejmuje następujące symbole terminalne:

- wyrazy funkcyjne: `"if", "and", "&&", "or", "||","not", "!", "else", "sub", "foreach", "return", "among"`
- referencje do węzłów drzewa języka docelowego (angielskiego) – napisy zaczynające się od znaku ‘%’ lub znaków ‘%%’
- referencje do węzłów drzewa wyrażenia j. źródłowego (polskiego) – napisy zaczynające się od znaku ‘@’,
- zmienne o wartościach nazw węzłów drzewa wyrażenia j. docelowego (angielskiego) – napisy zaczynające się od znaku ‘$’,
- atomy alfanumeryczne i liczby (określające wartości atrybutów, np. cech gramatycznych),
- atrybuty (określające nazwy atrybutów),
- symbole operatorów: `"!=", "==", ":=", "->","=>", "::", "(",")", "{",, }", "[", "]"`

W dodatku 1. zamieszczony jest fragment pliku tekstowego, który stanowi dane wejściowe programu *flex*.

Reguły systemu POLENG zapisane są w postaci ciągu instrukcji. Instrukcja języka reguł może mieć postać:

- instrukcji pustej
- instrukcji złożonej (w postaci ciągu instrukcji zamkniętych nawiasami klamrowymi)
- instrukcji prostej (np. instrukcja przypisania)
- instrukcji warunkowej *if* (o składni zgodnej z językiem C)
- instrukcji warunkowej *if, else*
- instrukcji transferu (stosowanej rekurencyjnie w regułach transferu)
- wywołania procedury z parametrami (stosowanej w regułach modyfikacji i syntezy)
- instrukcji FOREACH (stosowanej w regułach modyfikacji w celu wykonywania instrukcji dla wszystkich podwęzłów danego węzła)
- instrukcji przypisania referencji
- instrukcji zwrócenia wyniku przez podprogram.

Fragment pliku wejściowego do programu *bison*, opisujący gramatykę języka reguł podany jest w dodatku 2.

4. Reguły transferu

Reguły transferu odpowiadają za przekształcenie drzewa składniowego wyrażenia polskiego w odpowiadające mu drzewo wyrażenia angielskiego. W szczególności reguły transferu umożliwiają następujące przekształcenia:

- zmiana nazwy węzła nieterminalnego, np.

FR => NP (węzeł drzewa polskiego o nazwie *FR* „przechodzi" na węzeł o nazwie *NP*),

- zmiana nazwy węzła terminalnego, np.

'to' => 'it_is' (liść drzewa polskiego o nazwie *to* „przechodzi" na liść o nazwie *it_is*)

- zmiana etykiety gałęzi, np.

*:dop1 => :compl1 (*gałąź o etykiecie *dop1* transferowana jest na gałąź o etykiecie *compl1*)

- ustalenie wartości atrybutów cech gramatycznych, np.

Num:= @.L (jako wartość atrybutu *Num* podstaw wartość atrybutu *L* węzła drzewa źródłowego),

if (@.A = dk) {A := perf;} (jeśli wartością atrybutu *A* węzła drzewa źródłowego było *dk*, to za wartość atrybutu *A* węzła drzewa docelowego podstaw *perf*).

- ustalenie wartości atrybutów cech składniowych, np.

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

(Powyższa reguła umożliwia transfer zdań podrzędnych stanowiących oczekiwanie akomodacyjne czasownika – zgodnie z informacją o składni oczekiwań akomodacyjnych czasownika zaczerpniętą ze słownika, patrz [8]).

- usunięcie węzłów, które nie mają swojego odpowiednika, np. *delete(,oneself,)* (węzeł zaimka zwrotnego 'się' jest „wstępnie" usuwany, gdyż nie zachodzi odpowiedniość pomiędzy zwrotnością czasowników polskich i angielskich)
- dołączenie nowej gałęzi, np.

if (Refl) {insert_right(,'oneself',);} (jeśli czasownik angielski ma własność zwrotności, to do drzewa dołączana jest gałąź z zaimkiem zwrotnym)

- dołączenie nowego węzła, np.

$NomPron := insert_left(,NomPron, subject) (wygenerowanie odpowiednika podmiotu domyślnego)

- zmiana etykiety, np.

change_label(,compl1, subject) (taka sytuacja ma miejsce na przykład wtedy, gdy fraza rzeczownikowa pełniąca funkcję dopełnienia w zdaniu polskim ma spełniać funkcję podmiotu w zdaniu angielskim)

- wymiana etykiet, np.

exchange_labels(,compl1, subject) (dotyczy konstrukcji, w których podmiot i dopełnienie wymieniają się rolami, jak np. w zdaniu *Podobasz mi się – I like you*).

5. Reguły modyfikacji składniowej

Reguły modyfikacji składniowej mają na celu zmodyfikowanie drzewa otrzymanego po fazie transferu – zgodnie z zasadami składni języka angielskiego. Reguły modyfikacji są niezależne od reguł transferu – wydaje się, że mogłyby zostać wykorzystane w systemach tłumaczenia na język angielski z innych języków niż polski.

Przykłady reguł modyfikacji składniowej:

```
sub VP::default()
{
%Subject [Label = subject];
...

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

Powyższa reguła ustala pożądaną kolejność poddrzew drzewa o korzeniu w węźle *VP* (np. podmiot reprezentowany przez węzeł *%Subject* ma poprzedzać orzeczenie reprezentowane przez węzeł *%Head*). (Zauważmy, że

w regule tej instrukcja porządku poprzedzona jest zestawem instrukcji, które umożliwiają przypisanie referencji odpowiednim węzłom. Przykładowo, referencja *%Subject* przypisana zostaje do węzła, do którego prowadzi gałąź o etykiecie *subject*.)

```
Num := %Subject.Num;
%Head.Num := %Subject.Num;
```

Powyższa reguła dopasowuje liczbę orzeczenia do liczby podmiotu. (Reguła ta stosowana jest do węzła typu VP – reprezentującego frazę czasownikową. Składa się z dwóch przypisań – pierwsze nadaje odpowiednią wartość atrybutowi *Num* dla aktualnego węzła, zaś drugie dokonuje takiego samego przypisania dla atrybutu *Num* podwęzła %Head, który reprezentuje czasownik główny frazy.)

```
%Oneself.Num := %Subject.Num;
%Oneself.Pers := %Subject.Pers;
%Oneself.S := %Subject.S;
```

Powyższa reguła dopasowuje własności zaimka 'oneself' do podmiotu.

```
%AdvClause [Cat = Adv, find(,SubC,)];
if (Tense = past or Tense = past_perfect or Tense =
past_continuous or Tense = past_perfect_continous)
   {
      if (%SubC.Type = ob or %SubC.Type = that)
      {
         %SubC->dependent_speech();
      }
      AdvClause->dependent_speech();
   }
}
```

Ta reguła obsługuje zjawisko przesunięcia czasu dla określonych typów zdań podrzędnych w mowie zależnej.

6. Reguły syntezy morfologicznej

Reguły syntezy morfologicznej mają na celu wygenerowanie form fleksyjnych wyrazów języka angielskiego. Język opisu reguł zawiera wyrażenia regularne. Umożliwiają one formalny zapis reguł regularnej odmiany wyrazów języka angielskiego, np.

```
sub ::continuize(lex)
{
  if($lex =~ /[^eiyo]e$/)
     {
       $lex := substring($lex, -1) + 'ing';
```

```
      #agreeing, dying, hoeing
    }
  else if($lex =~ /ie$/)
    {
      $lex := substring($lex, -2) + 'ying';
      #dying, lying
    }
  else if($lex =~ /ic$/)
    {
      $lex := $lex + 'king';
      #panicking, picnicking;
    }
  else
    {
      $lex := $lex + 'ing';
    }
  return $lex;
}
```

Powyższa reguła opisuje sposób regularnego tworzenia formy ciągłej czasowników angielskich w zależności od końcówki formy podstawowej
Wyrażenia regularne stosowane są również do realizacji odmiany fleksyjnej wyrazów nieregularnych, które są oznaczane specjalnym kodem w słowniku systemu POLENG. Na przykład czasownik *brake* oznaczony jest w słowniku POLENG kodem *V3oke3oken*. Kod ten umożliwia wygenerowanie obu form imiesłowów (*broke, broken)* z formy podstawowej poprzez odcięcie z końca formy podstawowej ilości liter zapisanej w kodzie (tutaj odpowiednio: 3 i 3) i dołączenie zapisanych w kodzie sufixów (tutaj odpowiednio: *oke* i *oken*).

```
...
if EngInflection =~ /^V(\d)([a-z]*)(\d)([a-z]*)$/)
  {
    if(past_simple($tense, Stat))
    {
      $Form := substring($word, -$1) + $2;
    }
...
```

Powyższa reguła jest formalnym zapisem opisanych czynności odcinania i konkatenacji przy pomocy wyrażeń regularnych.

7. Podsumowanie

W pracy przedstawiono koncepcję modularnego tłumaczenia automatycznego z języka polskiego na język angielski w oparciu o reguły. Wydaje się, że rozbicie procesu translacji na niezależne moduły zwiększy możliwość kontroli poprawności reguł. Zaimplementowanie wszystkich

modułów w jednym – imperatywnym – języku programowania (C++) powinno poprawić efektywność działania systemu.

Bibliografia

[1] Lison M., (2002), w: Speech and Language Technology. Volume 6, Poznań.

[2] Graliński F., (2002), w: Speech and Language Technology. Volume 6, Poznań.

[3] Jassem K. (1998), *Struktura systemu tłumaczenia z języka polskiego na język angielski,* w: Speech and Language Technology. Volume 2, Poznań.

[4] Jassem K. (2000), *Dealing with Free Order and Non-Language Markers in a Top-Down Left First Algorithm,* w: Speech and Language Technology. Volume 4, Poznań.

[5] Jassem K. Lison M. (2001) *Classification, Storage and Processsing of lexical phrases in Polish-English Machine Translation*, w: Demenko, Puppel (red.) *PROSODY 2000, Proceeedings of ISKA International Workshop, PROSODY 2000, Kraków, 2 – 5 October 2000, Wydawnictwo UAM, Poznań, 2001*

[6] Grygiel K. (2000), *Kompilatory*, w miesięczniku Linux+, 2000/5, wyd. Software

[7] www.gnu.org

[8] Graliński F., Jassem K. (2001), *Formalizm opisu haseł w słowniku systemu POLENG – raport,* w: w: Speech and Language Technology. Volume 5, Poznań.

Dodatek 1. Fragment pliku wejściowego do programu *flex*. W pliku zdefiniowane są tokeny, czyli dopuszczalne symbole terminalne gramatyki opisu reguł.

```
PLUPPER [A-ZĄĆĘŁŃÓŚŹŻ]
PLLOWER [a-ząćęłń óśźż]
PLLETTER [a-ząćęłń óśźżA-ZĄĆĘŁŃÓŚŹŻ]

"if" return TTOKEN_IF;
("and"|"&&")   return TTOKEN_AND;
("or"|"||")    return TTOKEN_OR;
("not"|"!")    return TTOKEN_NOT;
"else" return TTOKEN_ELSE;
"sub" return TTOKEN_SUB;
"foreach" return TTOKEN_FOREACH;
```

```
"return" return TTOKEN_RETURN;
"among" return TTOKEN_AMONG;

   /* variable */
\${PLLETTER}({PLLETTER}|[_0-9])*
{ trparserlval.s = new string(trlextext+1);
    return TTOKEN_VAR; }

   /* plref */
\@{PLLETTER}({PLLETTER}|[_0-9])*
{ trparserlval.s = new string(trlextext+1);
    return TTOKEN_PLREF; }
```

Dodatek 2. Fragment pliku wejściowego do programu BISON. W pliku zdefiniowane są dopuszczalne instrukcje języka opisu reguł

```
alls: script
{
    trparser_all = $1;
}

script: instr
{
    $$ = new TScript;
    $$->push_back($1);
}
| script instr
{
    $$->push_back($2);
}

expr: TTOKEN_NUMBER
{
    $$ = new TRuleExpression(TRuleExpression::NUMBER);
    $$->n = $1;
}
| TTOKEN_ATOM
{
    $$ = new TRuleExpression(TRuleExpression::ATOM);
    $$->s = $1;
}
| TTOKEN_ATTRIBUTE
{
    $$ = new
TRuleExpression(TRuleExpression::ATTRIBUTE);
```

```
    $$->s = $1;
}
| TTOKEN_PLREF
{
    $$ = new TRuleExpression(TRuleExpression::PLREF);
    $$->s = $1;
}
| TTOKEN_VAR
{
    $$ = new TRuleExpression(TRuleExpression::VARIABLE);
    $$->s = $1;
}
...
}
| expr TTOKEN_OR expr
{
    $$ = new TRuleExpression(TRuleExpression::SUBR);
    $$->n = int(TRuleExpression::OPERATOR_OR);
    $$->subexprs = new vector<TRuleExpression*>;
    $$->subexprs->push_back($1);
    $$->subexprs->push_back($3);
}
```