Árbol ancestral *

*Note: Sub-titles are not captured in Xplore and should not be used

1st Peña Andia Victor Angelo
Ciencia de la computación
UTEC
Lima, Peru
victor.pena@utec.edu.pe

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—En este proyecto que se implemento en c++ un traductor de términos ancestrales. Para esto se creo una gramática capaz de detectar cada una de las palabras y poder luego traducirlas.**

***Index Terms*—gramática, C++, traductor, términos ancestrales**

I. Introduction

Los lenguages son un conjunto de simbolos definidos por $\sum$, donde con los elementos en sigma es posible crear cadenas de simbolos finitas. Pero, para conocer si una determinada oracion o conjunto de palabras estan formadas correctamente se usa el concepto de gramática. Este concepto ayuda a definir correctas cadenas que puedan ser reconocidas por un automata.

Estos conceptos ayudan a definir un determinada maquina capaz de resolver uno o unos ciertos problemas, dependiendo de sus estados.

Para este problema se uso los conceptos anteriores para si conseguir que una cadena de palabras sea reconocida. Luego, esta cadenas sea inspeccionada si es correcta. Finalmente, si es correcta se muestra la traducción de la frase, caso contrario se presenta un error y se interrumpe el proceso.

Por otro lado, se creo una representación intermedia que expresa la relación de padre o madre respecto a un función.

II. Planteamiento del problema

Para resolver el problema existe un conjunto infinito A1 que tiene los elementos ancestrales a traducir $A1 = \{mother, father, grandmother, greatgrandfather, ...\}$, dentro de este conjunto se identifico las palabras con las que la gramática trabaja: father, mother, great y grand.

Con estos datos se analizo la formación de una traducción intermedia de tipo: $mother = mo()ofather = fa()$ o, $grandmother = g(mo())$.

Al tener las palabras pertenecientes a la gramática y la conversión de palabra a representación intermedia, es posible disear la gramática adecuada para este lenguaje. La gramática diseada seria de este modo:

$$S- > GM|GF$$

$$G- > RD$$

$$R- > RA$$

$$F- > father$$

$$M- > mother$$

$$D- > grand$$

$$A- > great$$

Al tener las reglas por las cuales la gramática se forma es posible comenzar de derecha a izquierda a analizar la palabra.

En la segunda parte del problema se pide analizar la respuesta de la primera parte y traducirlo. Para esto la función debe hacer uso de una gramática similar a la usada para los imput's

$$S- > GM|GV$$

$$G- > RD$$

$$R- > RA$$

$$V- > vatter$$

$$M- > mutter$$

$$D- > gross$$

$$A- > ur$$

Con la gramatica anterior y el resultado de la funcion se logro traducir la frase y encontrar una representacion intermedia.

Finalmente, se recibe una oracion cullas palabras estan dentro del conjunto A3, siendo $A3 = \{The, mother, father, of, Mary, John\}$. Esta oracion pasara traducirce al lenguage A4, donde $A4 = \{Die, ein, mutter, vatter, von, Maria, John\}$

La gramatica del lenguage A3 es:

$$Sentence- > InitSentence2$$

$$Sentence2- > SecondSentence2|ONombre$$

$$Nombre- > Mary|John$$

$$Init- > Themother|Thefather$$

$$Second- > ofthemother|ofthefather$$

$$O- > of$$

La gramatica del lenguage A4 es:

$$Sentence2- > Second.Sentence2|GONombre$$

$$Nombre- > Mary|John$$

$$Second- > ur$$

$$O- > von$$

$$G- > gross$$

Con estas gramaticas creadas solo es necesario realizar un codigo que al leer las frases verifique que sean correctas, mientras se encarga de traducirlas.

III. Resolucion

El problema lo dividimos en dos pricipales partes, donde la primera se usa dos clases y en la segunda dos funciones. Se usa la funcion

$$string\ middleInterpretation(string\ word)$$

donde recibe como parametro $word$ que es la palabra que pasara a ser una funcion.

```
// functions.cpp
string middleInterpretation(string word){
  bool controlFoM = false, controlG = false;
  string partialWord;
  string middle;
  for (auto letter = word.rbegin(); letter !=
      word.rend() ; ++letter) {
    partialWord.insert(partialWord.begin(),*letter);
    if (partialWord == "father" and
        !controlFoM){
      middle.insert(0,"fa(");
      partialWord = "";
      controlFoM = true;
    }else if(partialWord == "mother" and
        !controlFoM){
      middle.insert(0,"mo(");
      partialWord = "";
      controlFoM = true;
    } else if(partialWord == "grand" and
        !controlG){
      middle.insert(0,"g(");
      middle.insert(middle.end(),')');
      partialWord = "";
      controlG = true;
    }else if(partialWord == "great" and
        controlG){
      middle.insert(0,"g(");
      middle.insert(middle.end(),')');
      partialWord = "";
    }else{
      if(partialWord.size() > 6 ){
        cout<<"Error en la frase:
            "<<partialWord<<endl;
        return "error";
      }
    }
  }
  if(!partialWord.empty()){
    cout<<"Error en la frase:
        "<<partialWord<<endl;
    return "error";
  }
  return middle;
}
```

En esta funcion, la variable locales controFoM se encarga de controlar si la frase ya recibio ”father” o ”mother”, ya que esto se debe recibir solo una vez. Por otro lado, controlG se encarga de asegurarse que se reciba solo una vez ”grand” y los demas sean ”great”. Luego, partialWord es la palabra que se forma concatenando cada caracter de derecha a izquierda. Los casos posibles en que la palabra parcial sea distinta a lo esperado se retorna ”error”. Si todo finalizaba como lo esperaba se devolvia la representacion intermedia.

Para la segunda parte del problema se uso una funcion que recibia la representacion intermedia y luego la traducia a aleman. Para esto se uso la funcion:

$$string\ finalInterpretation(string\ word)$$

. La funcion recibe el string $word$ que es la representacion intermedia a traducir.

```
//functions.cpp
string finalInterpretation(string word){
int count = 0;
bool controlFoM = false, controlG = false;
string partialWord;
string middle;
for (auto letter = word.rbegin(); letter !=
    word.rend() ; ++letter) {
  partialWord.insert(partialWord.begin(),
      *letter);
if (partialWord == ")") {
count += 1;
partialWord = "";
}else if (partialWord == "fa(" and
    !controlFoM and count>0){
middle.insert(0,"vatter");
partialWord = "";
count -=1;
controlFoM = true;
}else if(partialWord == "mo(" and !controlFoM
    and count>0){
middle.insert(0,"mutter");
partialWord = "";
controlFoM = true;
count -=1;
} else if( partialWord == "g(" and controlFoM
    and !controlG and count>0){
middle.insert(0,"gross");
partialWord = "";
controlG = true;
count -=1;
}else if (partialWord == "g(" and controlFoM
    and controlG and count>0){
middle.insert(0,"ur");
partialWord = "";
```

```
count -=1;
}else{
if(partialWord.size() > 3 or count == 0){
cout<<"Error en la frase:
    "<<partialWord<<endl;
return "error";
       }
     }
   }
return middle;
}
```

Las variables boleanas se usarn para controlar que la funcion input sea la adecuada. En caso no cumpla se retorna error. Por otro lado, la variable count sirve para tener en cuenta la cantidad de ")" para controlar la cantidad de funciones recibidas. Finalmente, al igual que la funcion anterior se itera de derecha a izquierda para obtener las funciones necesarias. El resultado sera la traduccion final en caso cumpla todos los requisistos o un mensaje de error.

La parte final del proyecto se uso las mismas tecnicas, la principal diferencia es el uso de una clase adicional. La primera clase, dictionary es una estructura que nos almacena las palabras en ingles. Su unica funcion nos permite obtener si la palabra esta dentro de dictonary. Por otro lado, la clase parse se usa para contener las palabras del otro lenguage y usarlas para contruir la frase traducida.

```
//parse.cpp
string parser::parse(string
    sentenceToTranslate) {
string translatedSentence;
string partialWord;
bool theControl = false;
bool nameControl = false;
bool ofControl = false;
bool control = false; //controls the first
    "of"
bool isFather = false; //controls if the last
    word read is father, else is mother
int count = 0;    //count the number of
    "father" or "mother" read
for (auto letter =
    sentenceToTranslate.rbegin();letter !=
    sentenceToTranslate.rend(); letter++ ) {
if(*letter != ' '){
partialWord.insert(partialWord.begin(),
    *letter);
}
if
    (wordsToTranslate->isInDictionary(partialWord)){
if ((partialWord == "Mary" or partialWord ==
    "John") and !nameControl and !theControl){
translatedSentence.insert(0, partialWord);
partialWord = "";
nameControl = true;
} else if( partialWord == "of" and
    nameControl and !ofControl ){
if (!control and !theControl){
translatedSentence.insert(0,translatedWords->at(6)+"
    ");
partialWord = "";
control = true;
ofControl = true;
theControl = false;
}else {
partialWord = "";
ofControl = true;
theControl = false;
}
} else if((partialWord == "father" or
    partialWord == "mother") and nameControl
    and ofControl){
count +=1;
isFather = partialWord == "father";
partialWord = "";
ofControl = false;
} else if (partialWord == "the" and
    !theControl){
partialWord = "";
theControl = true;
} else if(partialWord == "The"){
string grossurMaker;
if (isFather){
grossurMaker.insert(0,"vater ");
} else{
grossurMaker.insert(0,"mutter ");
}
for (int contador = 0; contador < count-1 ;
    ++contador) {
if(contador == 0){
grossurMaker.insert(0, "gross");
} else{
grossurMaker.insert(0,"ur");
}
}
translatedSentence.insert(0,grossurMaker);
if(count==1){
translatedSentence.insert(0,"Die ");
} else{
translatedSentence.insert(0,"Ein ");
}
partialWord ="";
}else{
if(partialWord.size() > 7){
cout<<"Error en la frase:
    "<<partialWord<<endl;
return "error";
}
}
}else{
if(partialWord.size() > 7){
cout<<"Error en la frase:
    "<<partialWord<<endl;
return "error";
}
}
if(!partialWord.empty()){
return "error";
}
return translatedSentence;
}
```

En la funcion presentada se uso variables boolenanas que funcionan cambiando de valor para aceptar la combinacion adecuada de palabras, son la manera de aceptar o rechazar lo que cumpla o incumpla la gramatica.

IV. Concluciones

En conclusion, si es posible disear la gramatica de un lenguage es posible disear un programa capaz de reconocerlo. La dificulta empieza cuando la gramatica es mas grande y el lenguage puede generar ambiguedades. Es por esto que en un inicio del problema se hizo la gramatica correspondiente para asi tener una guia para la resolucion del problema.