

TransLog

Oscar Araya Garbanzo - 2018002998
Saymon Astúa Madrigal - 2018143188

I Semestre
Año 2020

Lenguajes, compiladores e intérpretes

Área académica de Ingeniería en Computadores

Instituto Tecnológico de Costa Rica

TEC | Tecnológico
de Costa Rica

1.1 Descripción de los hechos y reglas implementadas.

Primeramente, y con la finalidad de facilitar el entendimiento de las reglas a describir a continuación, aquí se presenta la estructura y funcionamiento de cada uno de los hechos almacenados en la base de datos:

a) Sustantivo.

`sustantivo(Numero, Genero, PalabraEspanol, PalabraIngles).`

Indica si el sujeto es plural o singular

Indica si el sujeto es femenino o masculino (según se considere en el idioma español)

Palabra en español, en forma ["Perro"]

Palabra en inglés, en forma ["Dog"]

`sustantivo(plural, masculino, ["lenguajes"], ["languages"]).`

b) Verbo.

`verbo(Numero, Tiempo, Persona, PalabraEspanol, PalabraIngles).`

Indica si el verbo es plural o singular

Indica si el verbo es pasado, presente o futuro.

Indica si el verbo es en primera, segunda o tercera persona

Palabra en español, en forma ["Corre"]

Palabra en inglés, en forma ["Run"]

`verbo(singular, presente, tercera, ["come"], ["eats"]).`

c) Pronombre.

`pronombre(Numero, Persona, PalabraEspanol, PalabraIngles).`

Indica si el pronombre es plural o singular

Indica si el pronombre es en primera, segunda o tercera persona

Palabra en español, en forma ["Ellos"]

Palabra en inglés, en forma ["They"]

`pronombre(plural, primera, ["nosotros"], ["we"]).`

d) Adjetivo.

`adjetivo(Numero, Genero, PalabraEspanol, PalabraIngles).`

Indica si el
adjetivo es
plural o singular

Indica si el adjetivo
es masculino o femenino
(según se considere en
el idioma español)

Palabra en español,
en forma ["Rojo"]

Palabra en inglés,
en forma ["Red"]

`adjetivo(singular, masculino, ["rojo"], ["red"]).`

e) El resto de hechos: adverbio, preposición, interrogativo, conjunción y saludo, solo se componen de la **PalabraEspanol** y la **PalabraIngles**.

`adverbio(["comunmente"], ["commonly"]).`

`preposicion(["de"], ["of"]).`

`interrogativo(["Quiéenes"], ["Who"]).`

`conjuncion(["y"], ["and"]).`

`saludo(["Hola"], ["Hello"]).`

Estos últimos cinco tipos de hechos comparten su aridez, más no su nombre, lo que facilita la validación de las estructuras gramaticales. Además, es importante recalcar que, debido a que Prolog por defecto no permite la utilización de tildes y demás caracteres de la lengua española, se ha decidido que, para colocar una tilde, se deben colocar dos vocales consecutivas que representan esa misma vocal tildada. Por ejemplo:

Programación \equiv Programacioon

Último \equiv Uultimo

Raíz \equiv Raiiz

Al tratarse de un proyecto no demasiado extenso, se han utilizado pocas reglas que describen el funcionamiento y comportamiento del programa. A continuación, se detallará cada una de ellas, acompañada de una imagen de cómo luce en el código.

iniciarTraduccion: regla que inicia la ejecución del programa. Se encarga de dar la bienvenida al usuario en ambos idiomas, además de solicitar el **IdiomaSeleccionado** en consola. Una vez que el usuario coloca un idioma a traducir, se procede a llamar a la regla *traducir*.

```
iniciarTraduccion:-
    write(';Bienvenido a TransLog! Por favor, coloque el idioma del cual desea traducir. '),
    nl,
    write('Coloque "Es" para español y "In" para inglés. '),
    nl,
    write('Welcome to TransLog! Please, type the language that you want to traduce. '),
    nl,
    write('Type "Es" for spanish and "In" for english. '),
    nl,
    read(IdiomaSeleccionado),
    traducir(IdiomaSeleccionado),
    !.
```

traducir: regla que recibe como argumento al **IdiomaSeleccionado** y, en función de lo que se colocó en consola en la regla *iniciarTraduccion*, se notifica al usuario en el idioma en el que colocará el texto. Además, se solicita al usuario el **TextoIntroducido** en consola, para posteriormente separar el texto en una lista de palabras que se almacena en **TextoTraducir**. Con este **TextoTraducir** se llama a la regla *oracion*, que se encargará de la traducción como tal. Finalmente, la lista ya traducida **TextoTraducido** se concatena como un texto en la **RespuestaUsuario**, la cual es mostrada al usuario como la traducción finalizada. Adicional a esto, se colocó una última definición, que revisa si el usuario colocó un **IdiomaSeleccionado** que no sea “Es” ni “In”, e indica que se debe colocar nuevamente un idioma válido.

```
traducir(IdiomaSeleccionado):-
    IdiomaSeleccionado = "Es",
    write('Se ha seleccionado el idioma español. Por favor, coloque el texto a traducir: '),
    nl,
    read(TextoIntroducido),
    split_string(TextoIntroducido, " ", "Â;Â;", TextoTraducir),
    oracion(TextoTraducir, TextoTraducido, IdiomaSeleccionado),
    atomic_list_concat(TextoTraducido, ' ', RespuestaUsuario),
    write('El texto traducido de español a inglés es el siguiente: '),
    write(RespuestaUsuario).

traducir(IdiomaSeleccionado):-
    IdiomaSeleccionado = "In",
    write('You have selected the english language. Please, type the text that you want to traduce' '),
    nl,
    read(TextoIntroducido),
    split_string(TextoIntroducido, " ", "Â;Â;", TextoTraducir),
    oracion(TextoTraducido, TextoTraducir, IdiomaSeleccionado),
    atomic_list_concat(TextoTraducido, ' ', RespuestaUsuario),
    write('The traduced text from english to spanish is: '),
    write(RespuestaUsuario).

traducir(_):-
    write('Ha seleccionado un idioma no válido. Por favor, inténtelo nuevamente. '),
    nl,
    write('You have typed an incorrect language. Please, try it again. '),
    nl,
    nl,
    iniciarTraduccion.
```

concatenar: regla utilizada para concatenar los elementos de una lista. Por lo general, en TransLog se ha utilizado para conocer el restante de una lista, más que para concatenar como tal. De esta forma, es posible validar las traducciones dentro de las reglas posteriores, y generar estructuras válidas según la gramática establecida. La regla concatenar tiene tres definiciones distintas:

1. Recibe como argumentos a una lista vacía [], y dos **Lista** que deben ser iguales. Este es el caso de parada de la recursión.
2. Recibe como argumentos a una **Lista**, y dos listas que se “recortan” desde la **Cabeza**, estableciendo al **Resto** y **Resto2**. Esta definición recursiva continúa recortando la **Lista** hasta llegar al caso 1.
3. Recibe como argumentos a la **ListaA**, **ListaB**, **ListaC** y **ListaZ**. Su funcionamiento consiste en subdividir la tarea en dos *concatenar* de tipo 1 o 2, a partir de la **ListaX**.

```
concatenar([],Lista,Lista) .
```

```
concatenar([Cabeza|Resto], Lista, [Cabeza|Resto2]) :-  
    concatenar(Resto,Lista,Resto2) .
```

```
concatenar(ListaA, ListaB, ListaC, ListaZ) :-  
    concatenar(ListaA, ListaB, ListaX),  
    concatenar(ListaX, ListaC, ListaZ) .
```

sintagmaNominal: conjunto de reglas que describen el comportamiento esperado de un sintagma nominal. Su funcionamiento puede modelarse como la siguiente gramática libre de contexto:

$N \rightarrow s$

$N \rightarrow p$

$N \rightarrow a$

$N \rightarrow ds$

$N \rightarrow sa$

$N \rightarrow das$

$N \rightarrow dsa$

$N \rightarrow ab$

donde: s=sustantivo
p=pronombre
a=adjetivo
d=determinante
b=adverbio

se representan como terminales de la GLC, ya que a partir de estas producciones es que se llega al texto que contiene las palabras en español e inglés

Recibe como argumentos a la **ListaPalabrasEspanol**, la **ListaPalabrasIngles** y la **Persona**, que puede tomarse como variable anónima o no, dependiendo del contexto gramatical a evaluar. Se podría decir que el *sintagmaNominal* es la parte más atómica de la estructura de la oración, solo por detrás de las palabras individuales.

```
sintagmaNominal(PalabraEspanol, PalabraIngles, _):-
    sustantivo(_, _, PalabraEspanol, PalabraIngles).

sintagmaNominal(PalabraEspanol, PalabraIngles, PersonaConjugada):-
    pronombre(_, PersonaConjugada, PalabraEspanol, PalabraIngles).

sintagmaNominal(PalabraEspanol, PalabraIngles, _):-
    adjetivo(_, _, PalabraEspanol, PalabraIngles).

sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, Persona):-
    determinante(Número, Género, Persona, PalabraEspanol, PalabraIngles),
    sustantivo(Número, Género, PalabraEspanol2, PalabraIngles2),
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),
    concatenar(PalabraIngles, PalabraIngles2, ListaPalabrasIngles).

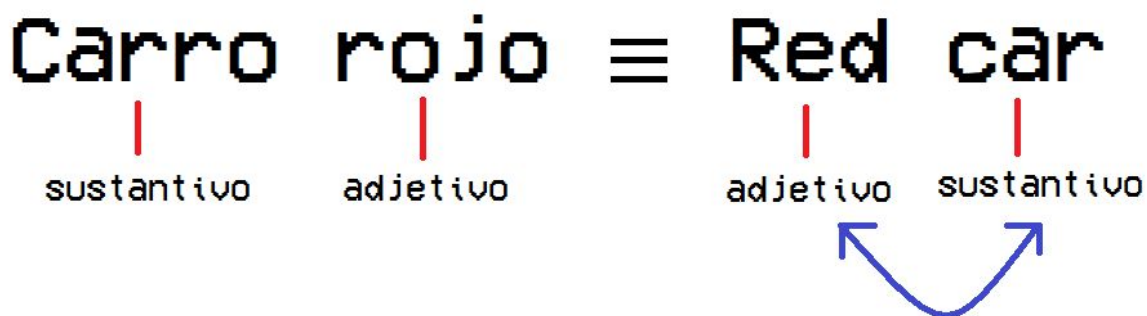
sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, _):-
    sustantivo(Número, Género, PalabraEspanol, PalabraIngles),
    adjetivo(Número, Género, PalabraEspanol2, PalabraIngles2),
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),
    concatenar(PalabraIngles2, PalabraIngles, ListaPalabrasIngles).

sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, Persona):-
    determinante(Número, Género, Persona, PalabraEspanol, PalabraIngles),
    adjetivo(Número, Género, PalabraEspanol2, PalabraIngles2),
    sustantivo(Número, Género, PalabraEspanol3, PalabraIngles3),
    concatenar(PalabraEspanol, PalabraEspanol2, PalabraEspanol3, ListaPalabrasEspanol),
    concatenar(PalabraIngles, PalabraIngles2, PalabraIngles3, ListaPalabrasIngles).

sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, Persona):-
    determinante(Número, Género, Persona, PalabraEspanol, PalabraIngles),
    sustantivo(Número, Género, PalabraEspanol2, PalabraIngles2),
    adjetivo(Número, Género, PalabraEspanol3, PalabraIngles3),
    concatenar(PalabraEspanol, PalabraEspanol2, PalabraEspanol3, ListaPalabrasEspanol),
    concatenar(PalabraIngles, PalabraIngles3, PalabraIngles2, ListaPalabrasIngles).

sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, _):-
    adjetivo(_, _, PalabraEspanol, PalabraIngles),
    adverbio(PalabraEspanol2, PalabraIngles2),
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),
    concatenar(PalabraIngles, PalabraIngles2, ListaPalabrasIngles).
```

El *sintagmaNominal* también es el encargado de tomar las palabras y modificar su orden, según se requiera en el idioma inglés o español. Por ejemplo:



```
concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),
concatenar(PalabraIngles2, PalabraIngles, ListaPalabrasIngles).
```

sintagmaVerbal: se vale del funcionamiento de *sintagmaNominal* para construir estructuras más complejas. Recibe como parámetros a la **PalabraEspanol**, **PalabraIngles**, y la **Persona**, además de la **ListaPalabrasEspanol** y la **ListaPalabrasIngles**. Esta regla evalúa si existe un verbo antes del *sintagmaNominal* o, en el caso más simple, evalúa si la **PalabraEspanol** o la **PalabraIngles** son un verbo almacenado en la base de datos de Prolog..

```
sintagmaVerbal(PalabraEspanol, PalabraIngles, Persona):-  
    verbo(_, _, Persona, PalabraEspanol, PalabraIngles).
```

```
sintagmaVerbal(ListaPalabrasEspanol, ListaPalabrasIngles, Persona):-  
    verbo(_, _, Persona, PalabraEspanol, PalabraIngles),  
    sintagmaNominal(PalabraEspanol2, PalabraIngles2, Persona),  
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),  
    concatenar(PalabraIngles, PalabraIngles2, ListaPalabrasIngles).
```

oracionSimple: conjunto de reglas que construye estructuras aún más complejas que *sintagmaVerbal* y *sintagmaNominal*. Recibe como argumentos a la **ListaPalabrasEspanol** y la **ListaPalabrasIngles**. Revisa si la *oracionSimple* está construida de tres formas posibles:

1. Se trata de un saludo. Utilizado como prueba para únicamente traducir una palabra, por ejemplo, “Hola” como “Hello”.
2. Se trata de un *sintagmaNominal*.
3. Se trata de un *sintagmaNominal* acompañado de un *sintagmaVerbal*.
4. Se trata de un *interrogativo* (pregunta tipo “¿Cómo?” o “How?”) acompañado de un *sintagmaVerbal*.

```
oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-  
    saludo(PalabraEspanol, PalabraIngles),  
    concatenar(PalabraEspanol, ["!"], ListaPalabrasEspanol),  
    concatenar(PalabraIngles, ["!"], ListaPalabrasIngles).
```

```
oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-  
    sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, _).
```

```
oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-  
    sintagmaNominal(PalabraEspanol, PalabraIngles, Persona),  
    sintagmaVerbal(PalabraEspanol2, PalabraIngles2, Persona),  
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),  
    concatenar(PalabraIngles, PalabraIngles2, ListaPalabrasIngles).
```

```
oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-  
    interrogativo(PalabraEspanol, PalabraIngles),  
    sintagmaVerbal(PalabraEspanol2, PalabraIngles2, _),  
    concatenar(PalabraEspanol, PalabraEspanol2, CabezaListaPalabrasEspanol),  
    concatenar(CabezaListaPalabrasEspanol, ["?"], ListaPalabrasEspanol),  
    concatenar(PalabraIngles, PalabraIngles2, CabezaListaPalabrasIngles),  
    concatenar(CabezaListaPalabrasIngles, ["?"], ListaPalabrasIngles).
```

Retomando nuevamente la notación de las gramáticas libres de contexto, la definición de *oracionSimple* puede representarse como:

S	→	h	
S	→	N	donde: S=oración simple
S	→	NU	h=saludo
			N=sintagma nominal
			U=sintagma verbal
S	→	qU	q=interrogativo

oracion: regla que recibe la lista de **PalabrasEspanol** y la lista de **PalabrasIngles** como argumentos, además del **IdiomaSeleccionado**. Esta regla es la base de la traducción en TransLog, al ser en sí el inicio de la gramática libre de contexto. Se decidió establecer que las oraciones tengan tres posibles estructuras:

1. Únicamente compuestas por una *oracionSimple*.
2. Compuestas de una *oracionSimple*, una *conjuncion* y nuevamente una *oracion*.
3. Compuestas de una *oracionSimple*, una *preposicion* y nuevamente una *oracion*.

```
oracion(PalabrasEspanol, PalabrasIngles, _):-
    oracionSimple(PalabrasEspanol, PalabrasIngles).
```

```
oracion(PalabrasEspanol, PalabrasIngles, _):-
    oracionSimple(E1, I1),
    conjuncion(E2, I2),
    concatenar(E1, E2, E3),
    concatenar(I1, I2, I3),
    concatenar(E3, E4, PalabrasEspanol),
    concatenar(I3, I4, PalabrasIngles),
    oracion(E4, I4, _),
    concatenar(E3, E4, PalabrasEspanol),
    concatenar(I3, I4, PalabrasIngles).
```

```
oracion(PalabrasEspanol, PalabrasIngles, _):-
    oracionSimple(E1, I1),
    preposicion(E2, I2),
    concatenar(E1, E2, E3),
    concatenar(I1, I2, I3),
    concatenar(E3, E4, PalabrasEspanol),
    concatenar(I3, I4, PalabrasIngles),
    oracion(E4, I4, _),
    concatenar(E3, E4, PalabrasEspanol),
    concatenar(I3, I4, PalabrasIngles).
```


Este comportamiento recursivo de “llamar” nuevamente a *oracion* se ajusta enormemente con el modo de funcionamiento de Prolog, y a su vez, con el comportamiento de las gramáticas libres de contexto. Si se expresan estas anteriores estructuras como una GLC, se tendría que:

$$\begin{aligned} O &\rightarrow S \\ O &\rightarrow ScO \\ O &\rightarrow SpO \\ O &\rightarrow e \end{aligned}$$

donde: O=oración
S=oración simple
c=conjunción
p=Preposición
e=error

Nótese que entre las producciones de oración se encuentra “error”, que es el nombre que se le ha dado al último caso. Esta última producción es una regla que se llama únicamente en el caso de que TransLog no sea capaz de identificar una estructura dentro de la oración, y se compone del siguiente mensaje al usuario:

```
oracion(_,_, IdiomaSeleccionado):-
    write('TransLog no ha sido capaz de reconocer de manera
    exitosa el texto introducido. Por favor, inténtelo de nuevo. '),
    nl,
    write('TransLog has not been capable to succesfully recon
    the typed text. Please, try it again. '),
    nl,
    nl,
    traducir(IdiomaSeleccionado).
```

Cabe recalcar que, gracias al comportamiento recursivo característico de Prolog, la interpretación de la gramática libre de contexto no es demasiado compleja. Todo el funcionamiento de la gramática de TransLog puede resumirse en:

$$\begin{aligned} O &\rightarrow S \mid ScO \mid SpO \mid e \\ S &\rightarrow h \mid N \mid NV \mid qU \\ U &\rightarrow v \mid vN \\ N &\rightarrow das \mid dsa \\ N &\rightarrow ds \mid sa \mid ab \\ N &\rightarrow s \mid p \mid a \end{aligned}$$

No terminales:

O=oración
S=oración simple
U=sintagma verbal
N=sintagma nominal

Terminales:

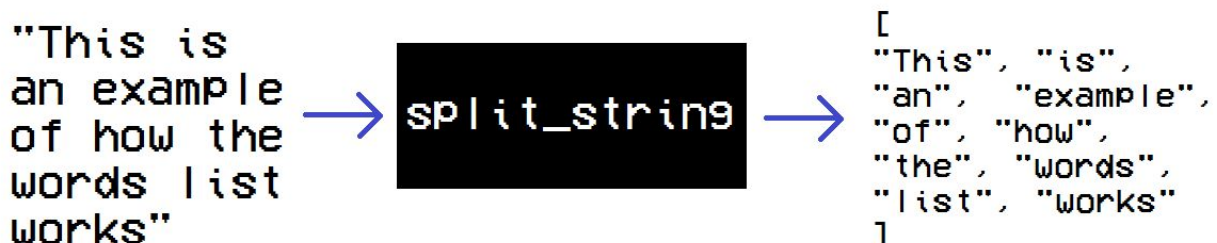
c=conjunción
p=Preposición
e=error
h=saludo
b=adverbio
q=interrogativo
d=determinante
a=adjetivo
s=sustantivo

1.2 Descripción de las estructuras de datos utilizadas.

ListaPalabrasEspanol / ListaPalabrasIngles: lista utilizada para contener a la oración colocada por el usuario, y la oración traducida por TransLog. Cual es cada una de ellas depende del IdiomaSeleccionado en la regla *iniciarTraduccion*.

Se decidió utilizar listas de este tipo debido a que es una forma sencilla de trabajar oraciones, al poder tomar cada palabra como una posición dentro de la lista.

Son creadas a partir de la función reservada *split_string*, dentro de la regla *traducir*. A partir de un texto plano entre comillas, se obtiene una lista de la siguiente forma:



```
"This is
an example
of how the
words list
works" → split_string → [
    "This", "is",
    "an", "example",
    "of", "how",
    "the", "words",
    "list", "works"
]
```

Además, el concatenar y agregar elementos de la oración original y traducción, respectivamente, es más sencillo, al únicamente tener que modificar posiciones de una lista, en vez de tratar con el string como tal. La modificación eficiente del orden de la gramática entre el idioma español e inglés es también posible al uso de estas listas.

PalabraEspanol / PalabraIngles: lista de un elemento que contiene únicamente a una palabra, ya sea en español o en inglés. Tienen dos utilidades principales. La primera es la de almacenar a las palabras al ser declaradas en los hechos, con la forma:

```
saludo(["Hola"], ["Hello"]).
```

Su segunda y más importante utilidad, y el motivo del porqué se almacenan las palabras en listas de un elemento en vez de strings corrientes, es que en la regla *concatenar*, en su condición de parada, resulta más sencillo el comparar listas que el estar accediendo a posiciones de la misma para revisar las palabras:

```
concatenar([], Lista, Lista).
```

Se puede decir que, la utilización de *PalabraEspanol* y *PalabraIngles* es una consecuencia de haber implementado la función *split_string*, al requerir de comparar y recorrer listas de una forma más directa y eficiente.

1.3 Descripción detallada del o los algoritmos desarrollados.

TransLog funciona a partir de un único algoritmo recursivo, compuesto de las reglas y hechos explicadas anteriormente. Su comportamiento puede ser modelado como una gramática libre de contexto, donde las producciones son las distintas cláusulas de las reglas de *oracion*, *oracionSimple*, *sintagmaVerbal* y *sintagmaNominal* (que a su vez representan a los no terminales).

Para facilitar el entendimiento de este algoritmo de traducción, resulta conveniente el uso de un ejemplo sencillo. Supongamos que se quiere traducir la frase “**El hombre come la manzana**”, del idioma español al idioma inglés.

Primero, el programa se encargaría de dividir a la oración en una lista con la forma **[“El”, “hombre”, “come”, “la”, manzana”]**, mediante la función reservada *split_string*, dentro de la regla *traducir*.

Seguidamente, TransLog revisa una por una las definiciones de el inicio del algoritmo, dentro de la regla *oracion*. Nótese que, dentro de las posibles formas que podría tener una *oracion*, ésta en particular solo cuenta con una *oracionSimple*, ya que no posee ni *conjuncion* o *preposicion*. Por el propio funcionamiento de backtracking de Prolog, se revisará la primera definición que se encuentre de la regla *oracion*, y aunque ya se sepa de antemano que si es la opción correcta, no hay que olvidar que el programa no sabe esto, y solamente está “tanteando”.

```
oracion(PalabrasEspanol, PalabrasIngles, _):-  
    oracionSimple(PalabrasEspanol, PalabrasIngles).
```

```
oracion(PalabrasEspanol, PalabrasIngles, _):-  
    oracionSimple(E1, I1),  
    conjuncion(E2, I2),  
    concatenar(E1, E2, E3),  
    concatenar(I1, I2, I3),  
    concatenar(E3, E4, PalabrasEspanol),  
    concatenar(I3, I4, PalabrasIngles),  
    oracion(E4, I4, _),  
    concatenar(E3, E4, PalabrasEspanol),  
    concatenar(I3, I4, PalabrasIngles).
```

```
oracion(PalabrasEspanol, PalabrasIngles, _):-  
    oracionSimple(E1, I1),  
    preposicion(E2, I2),  
    concatenar(E1, E2, E3),  
    concatenar(I1, I2, I3),  
    concatenar(E3, E4, PalabrasEspanol),  
    concatenar(I3, I4, PalabrasIngles),  
    oracion(E4, I4, _),  
    concatenar(E3, E4, PalabrasEspanol),  
    concatenar(I3, I4, PalabrasIngles).
```

A continuación, y al evaluar la validez de *oracionSimple*, se continúa con las definiciones de ésta última regla. La primera de ella describe un *saludo*, que según sus cláusulas debe ser encontrado en la base de datos a partir de **PalabraEspanol** y **PalabraIngles**. Revisando en las mismas cláusulas de *oracionSimple*, se ve que **PalabraEspanol** y **PalabraIngles** deben concordar con las obtenidas en *concatenar*, por lo que TransLog continúa evaluando dicha regla.

```
oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-
    saludo(PalabraEspanol, PalabraIngles),
    concatenar(PalabraEspanol, ["!"], ListaPalabrasEspanol),
    concatenar(PalabraIngles, ["!"], ListaPalabrasIngles).
```

```
oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-
    sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, _).
```

```
oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-
    sintagmaNominal(PalabraEspanol, PalabraIngles, Persona),
    sintagmaVerbal(PalabraEspanol2, PalabraIngles2, Persona),
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),
    concatenar(PalabraIngles, PalabraIngles2, ListaPalabrasIngles).
```

```
oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-
    interrogativo(PalabraEspanol, PalabraIngles),
    sintagmaVerbal(PalabraEspanol2, PalabraIngles2, _),
    concatenar(PalabraEspanol, PalabraEspanol2, CabezaListaPalabrasEspanol),
    concatenar(CabezaListaPalabrasEspanol, ["?"], ListaPalabrasEspanol),
    concatenar(PalabraIngles, PalabraIngles2, CabezaListaPalabrasIngles),
    concatenar(CabezaListaPalabrasIngles, ["?"], ListaPalabrasIngles).
```

La regla *concatenar* tiene tres definiciones, pero solo dos comparten aridad con el caso específico, por lo que la tercer inmediatamente puede ser descartada. La primera definición de *concatenar* indica que la **PalabraEspanol** debe ser vacía, además de que **ListaPalabrasEspanol** debe ser igual a ["!"]. Al esta última condición ser falsa, el programa revisa la siguiente definición, en búsqueda de alguna que sí cumpla con lo esperado. A este procedimiento de retroceder es a lo que se le llama backtracking.

```
concatenar([], Lista, Lista).
```

```
concatenar([Cabeza|Resto], Lista, [Cabeza|Resto2]):-
    concatenar(Resto, Lista, Resto2).
```

```
concatenar(ListaA, ListaB, ListaC, ListaZ) :-
    concatenar(ListaA, ListaB, ListaX),
    concatenar(ListaX, ListaC, ListaZ).
```


Al evaluar esta segunda definición, TransLog inicia un proceso recursivo hasta verificar cual es el último elemento de la **Lista**, y finalmente, compararlo con el elemento ["!"] en la definición descartada en el paso anterior. Cabe recalcar que este descarte ocurre numerosas veces, una por cada iteración recursiva.

```
concatenar([],Lista,Lista).
```

```
concatenar([Cabeza|Resto], Lista, [Cabeza|Resto2]):-  
    concatenar(Resto,Lista,Resto2).
```

```
concatenar(ListaA, ListaB, ListaC, ListaZ) :-  
    concatenar(ListaA, ListaB, ListaX),  
    concatenar(ListaX, ListaC, ListaZ).
```

El programa termina descubriendo que no hay forma de que el último elemento de la **Lista** sea ["!"], por lo que se tiene que descartar esta primera definición de *oracionSimple*. TransLog entonces retrocede, y revisa la siguiente definición.

```
oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-  
    saludo(PalabraEspanol,PalabraIngles),  
    concatenar(PalabraEspanol, ["!"], ListaPalabrasEspanol),  
    concatenar(PalabraIngles, ["!"], ListaPalabrasIngles).
```

```
oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-  
    sintagmaNominal(ListaPalabrasEspanol,ListaPalabrasIngles,_).
```

```
oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-  
    sintagmaNominal(PalabraEspanol, PalabraIngles, Persona),  
    sintagmaVerbal(PalabraEspanol2, PalabraIngles2, Persona),  
    concatenar(PalabraEspanol,PalabraEspanol2,ListaPalabrasEspanol),  
    concatenar(PalabraIngles,PalabraIngles2,ListaPalabrasIngles).
```

```
oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-  
    interrogativo(PalabraEspanol, PalabraIngles),  
    sintagmaVerbal(PalabraEspanol2, PalabraIngles2, _),  
    concatenar(PalabraEspanol, PalabraEspanol2, CabezaListaPalabrasEspanol),  
    concatenar(CabezaListaPalabrasEspanol, ["?"], ListaPalabrasEspanol),  
    concatenar(PalabraIngles, PalabraIngles2, CabezaListaPalabrasIngles),  
    concatenar(CabezaListaPalabrasIngles, ["?"], ListaPalabrasIngles).
```

Esta segunda definición indica que la *oracionSimple* es simplemente un *sintagmaNominal*, por lo que procede a validar las definiciones de este último. Las tres primeras definiciones pueden ser descartadas fácilmente. Aunque Prolog si las evalúe una por una, se sabe de antemano que ninguna es la correcta, ya que en este momento **PalabraEspanol** y **PalabraIngles** son listas de palabras, y no palabras individuales, por lo que no existe ningún *sustantivo*, *pronombre* o *adjetivo* que pueda

encajar. Por facilidad, serán descartadas de forma inmediata en esta explicación. Esto haría que Prolog evalúe la cuarta definición.

```
sintagmaNominal(PalabraEspanol, PalabraIngles, _):-  
    sustantivo(_, _, PalabraEspanol, PalabraIngles)  
  
sintagmaNominal(PalabraEspanol, PalabraIngles, PersonaConjugada):-  
    pronombre(_, PersonaConjugada, PalabraEspanol, PalabraIngles).  
  
sintagmaNominal(PalabraEspanol, PalabraIngles, _):-  
    adjetivo( , , PalabraEspanol, PalabraIngles).
```

```
sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, Persona):-  
    determinante(Numero, Genero, Persona, PalabraEspanol, PalabraIngles),  
    sustantivo(Numero, Genero, PalabraEspanol2, PalabraIngles2),  
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),  
    concatenar(PalabraIngles, PalabraIngles2, ListaPalabrasIngles).  
  
sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, _):-  
    sustantivo(Numero, Genero, PalabraEspanol, PalabraIngles),  
    adjetivo(Numero, Genero, PalabraEspanol2, PalabraIngles2),  
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),  
    concatenar(PalabraIngles2, PalabraIngles, ListaPalabrasIngles).  
  
sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, Persona):-  
    determinante(Numero, Genero, Persona, PalabraEspanol, PalabraIngles),  
    adjetivo(Numero, Genero, PalabraEspanol2, PalabraIngles2),  
    sustantivo(Numero, Genero, PalabraEspanol3, PalabraIngles3),  
    concatenar(PalabraEspanol, PalabraEspanol2, PalabraEspanol3, ListaPalabrasEspanol),  
    concatenar(PalabraIngles, PalabraIngles2, PalabraIngles3, ListaPalabrasIngles).  
  
sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, Persona):-  
    determinante(Numero, Genero, Persona, PalabraEspanol, PalabraIngles),  
    sustantivo(Numero, Genero, PalabraEspanol2, PalabraIngles2),  
    adjetivo(Numero, Genero, PalabraEspanol3, PalabraIngles3),  
    concatenar(PalabraEspanol, PalabraEspanol2, PalabraEspanol3, ListaPalabrasEspanol),  
    concatenar(PalabraIngles, PalabraIngles3, PalabraIngles2, ListaPalabrasIngles).  
  
sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, _):-  
    adjetivo(_, _, PalabraEspanol, PalabraIngles),  
    adverbio(PalabraEspanol2, PalabraIngles2),  
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),  
    concatenar(PalabraIngles, PalabraIngles2, ListaPalabrasIngles).
```

Esta cuarta definición de *sintagmaNominal* propone que la *oracionSimple* se trata de un *determinante* acompañado de un *sustantivo*, por lo que hace falta nuevamente conocer el valor de **PalabraEspanol**, **PalabraIngles**, **PalabraEspanol2** y **PalabraIngles2** por medio de *concatenar*. Repitiendo el comportamiento anteriormente explicado, el programa se entera de que, si bien sí existe un **determinante**(_, _, _, ["el"], ["the"]) que encaja con la primera posición de la **ListaPalabrasEspanol**, no existe un *sustantivo* que encaje con la estructura ["hombre", "come", "la", "manzana"], por lo que esta cuarta definición también debe ser descartada.


```

sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, Persona):-
    determinante(Numero, Genero, Persona, PalabraEspanol, PalabraIngles),
    sustantivo(Numero, Genero, PalabraEspanol2, PalabraIngles2),
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),
    concatenar(PalabraIngles, PalabraIngles2, ListaPalabrasIngles).

sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, _):-
    sustantivo(Numero, Genero, PalabraEspanol, PalabraIngles),
    adjetivo(Numero, Genero, PalabraEspanol2, PalabraIngles2),
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),
    concatenar(PalabraIngles2, PalabraIngles, ListaPalabrasIngles).

sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, Persona):-
    determinante(Numero, Genero, Persona, PalabraEspanol, PalabraIngles),
    adjetivo(Numero, Genero, PalabraEspanol2, PalabraIngles2),
    sustantivo(Numero, Genero, PalabraEspanol3, PalabraIngles3),
    concatenar(PalabraEspanol, PalabraEspanol2, PalabraEspanol3, ListaPalabrasEspanol),
    concatenar(PalabraIngles, PalabraIngles2, PalabraIngles3, ListaPalabrasIngles).

sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, Persona):-
    determinante(Numero, Genero, Persona, PalabraEspanol, PalabraIngles),
    sustantivo(Numero, Genero, PalabraEspanol2, PalabraIngles2),
    adjetivo(Numero, Genero, PalabraEspanol3, PalabraIngles3),
    concatenar(PalabraEspanol, PalabraEspanol2, PalabraEspanol3, ListaPalabrasEspanol),
    concatenar(PalabraIngles, PalabraIngles3, PalabraIngles2, ListaPalabrasIngles).

sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, _):-
    adjetivo(_, _, PalabraEspanol, PalabraIngles),
    adverbio(PalabraEspanol2, PalabraIngles2),
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),
    concatenar(PalabraIngles, PalabraIngles2, ListaPalabrasIngles).

```

No solo esta, ya que Prolog continúa evaluando todas las posibles definiciones de *sintagmaNominal*, hasta darse cuenta que ninguna puede encajar con la **ListaPalabrasEspanol**. Esto lleva al programa a la conclusión de que no se trata de una *oracionSimple* sólo formada por un *sintagmaNominal*, por lo que debe retroceder para revisar la tercera definición de *oracionSimple*.

```

oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-
    saludo(PalabraEspanol, PalabraIngles),
    concatenar(PalabraEspanol, ["!"], ListaPalabrasEspanol),
    concatenar(PalabraIngles, ["!"], ListaPalabrasIngles).

oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-
    sintagmaNominal(ListaPalabrasEspanol, ListaPalabrasIngles, _).

oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-
    sintagmaNominal(PalabraEspanol, PalabraIngles, Persona),
    sintagmaVerbal(PalabraEspanol2, PalabraIngles2, Persona),
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),
    concatenar(PalabraIngles, PalabraIngles2, ListaPalabrasIngles).

oracionSimple(ListaPalabrasEspanol, ListaPalabrasIngles):-
    interrogativo(PalabraEspanol, PalabraIngles),
    sintagmaVerbal(PalabraEspanol2, PalabraIngles2, _),
    concatenar(PalabraEspanol, PalabraEspanol2, CabezaListaPalabrasEspanol),
    concatenar(CabezaListaPalabrasEspanol, ["?"], ListaPalabrasEspanol),
    concatenar(PalabraIngles, PalabraIngles2, CabezaListaPalabrasIngles),
    concatenar(CabezaListaPalabrasIngles, ["?"], ListaPalabrasIngles).

```

Con esta tercera definición, TransLog encuentra que en efecto, la primera parte de la **ListaPalabrasEspanol** se trata de un *sintagmaNominal* de tipo determinante acompañado de sustantivo [“el”, “hombre”], por lo que se continúa verificando si el resto de la **ListaPalabrasEspanol** [“come”, “la”, “manzana”] encaja en alguna definición de *sintagmaVerbal*.

Dentro de *sintagmaVerbal* la primera definición es descartada, ya que no existe un verbo en la base de datos que concuerde con la **PalabraEspanol** dada por **["come", "la", "manzana"]**. Al evaluar la segunda definición, se encuentra con que, en efecto, el **verbo(, , , ["come"], ["eat"])** existe en la base de datos, por lo que solo restaría verificar si **["la", "manzana"]** puede encajar en un *sintagmaNominal*.

```
sintagmaVerbal(PalabraEspanol, PalabraIngles, Persona):-  
    verbo(, , Persona, PalabraEspanol, PalabraIngles).
```

```
sintagmaVerbal(ListaPalabrasEspanol, ListaPalabrasIngles, Persona):-  
    verbo(, , Persona, PalabraEspanol, PalabraIngles),  
    sintagmaNominal(PalabraEspanol2, PalabraIngles2, Persona),  
    concatenar(PalabraEspanol, PalabraEspanol2, ListaPalabrasEspanol),  
    concatenar(PalabraIngles, PalabraIngles2, ListaPalabrasIngles).
```

En *sintagmaNominal*, y repitiendo las iteraciones que fueron explicadas anteriormente, TransLog encuentra que **["la", "manzana"]** si es un *sintagmaNominal* válido, ya que la cuarta definición describe un *determinante* acompañado de un *sustantivo*. Como **determinante(, , , ["la"], ["the"])** y **sustantivo(, , , ["manzana"], ["apple"])** existen en la BD, se puede decir que se ha finalizado la validación de la estructura de la oración introducida por el usuario.

```
sintagma_nominal(L1, L2, Persona) :- determinante(Numero, Genero, Persona, X, XX),  
    sustantivo(Numero, Genero, Y, YY), adjetivo(Numero, Genero, Z, ZZ),  
    concatenar(X, Y, Z, L1), concatenar(XX, ZZ, YY, L2).
```

Este comportamiento de backtracking es el que hace posible el funcionamiento de TransLog. El programa intenta cada posible estructura y cada posible palabra en la base de datos hasta dar con un resultado correcto, y en caso de no encontrarlo, debido a limitaciones de cómo fué programado o de gramática, se notifica al usuario.

Este ejemplo explicado anteriormente es solo uno de casi infinitos casos en los que se puede desarrollar el algoritmo de traducción. El poder de Prolog está en que justamente esta funcionalidad recursiva le permite poder abarcar un enorme multitud de posibles funciones en pocas líneas de código.

Una vez que se da con un resultado correcto, TransLog concatena todas las posiciones de la ListaPalabrasIngles o la ListaPalabrasEspanol (según sea el idioma a traducir), y lo muestra al usuario de la siguiente manera:

```
Ingresa el texto:  
|: "el hombre come la manzana".  
the man eats the apple  
true.
```


1.4 Problemas sin solución.

- El programa no puede reconocer oraciones en español donde el sujeto esté intrínseco dentro de la misma, es decir, que no haya un sujeto escrito pero que por la conjugación del verbo, en español se sabe a cuál sujeto pertenece. En el caso de este programa no se logró implementar, por lo que hay que escribirle el sujeto de manera explícita. Ej: ¿Cómo estás? → ¿Cómo estás **tú**?
- El programa no reconoce oraciones demasiado extensas, pero por la naturaleza de la tarea, esto no es un problema tan grave.
- El programa no reconoce todas las combinaciones gramaticales posibles, ya que se podría explotar alguna carencia en su gramática libre de contexto para hacer que no pueda generar un resultado.

1.5 Plan de actividades.

Descripción	Responsable	Tiempo Aproximado	Fecha de entrega
Primera reunión. Lectura de la especificación y análisis de la misma. Recopilar dudas para preguntar al profesor. Asignar tareas.	Saymon y Oscar	1 hora	6 de julio
Investigar interacciones en prolog (Escribir en consola, mostrar datos en consola) y sobre las gramáticas libres de contexto	Saymon	3 horas	7 de julio
Investigar sobre la estructura o	Saymon	6 horas	11 de julio

gramática que tiene el idioma español y el inglés			
Crear la base de datos con los diferentes elementos de una oración, tanto en español, como en inglés. (verbos, sustantivos, determinantes, adjetivos, ...)	Saymon y Oscar	8 horas	12 de julio
Implementar en Prolog una gramática libre de contexto para saludos, preguntas y oraciones sencillas	Saymon y Oscar	8 horas	13 de julio
Crear la interfaz de interacción con el usuario	Saymon y Oscar	3 horas	15 de julio

1.6 Problemas encontrados

- En ocasiones el programa se metía a un bucle del cual no podía salir. Eso sucedía por gestionar mal la recursividad de prolog y al principio por no saber trabajar con este lenguaje. En otras ocasiones este bucle era provocado porque la base de datos era demasiado grande, por lo tanto el tiempo en realizar comparaciones y demás funciones propias del lenguaje era demasiado. En este caso si se podía salir del bucle, solo que si duraba demasiado tiempo. En este caso lo que se hace es reducir la base de datos con lo más utilizado.

1.7 Conclusiones y recomendaciones del proyecto.

Conclusiones:

- La gramática libre de contexto se implementó mediante el uso de concatenaciones en lugar de usar listas de inferencia. Se sabe que las listas de inferencia convierten el programa más eficiente, pero mediante el uso de las concatenaciones se vuelve más fácil de programar y entender el funcionamiento. También se usó concatenaciones porque al tener una base de datos limitada, entonces la eficiencia no era tan diferente si se usaba un método u otro.
- El uso de prolog para el procesamiento del lenguaje natural y la traducción de oraciones fue bastante ventajoso por las características propias del lenguaje, siendo el backtracking una de las más importantes, ya que el programador se desentiende en la implementación de este tipo de algoritmo y se puede enfocar mejor en la implementación de una gramática efectiva.

Recomendaciones:

- Es importante conocer el paradigma lógico de programación y las funciones específicas que ofrece Prolog como lenguaje de programación para facilitar la creación del traductor como tal. Lo que se puede hacer es practicar bastante y hacer muchas pruebas para lograr entenderlo mejor y así programar de manera más eficiente.
- Tener una base de datos más amplia permite tener un alcance mayor en la traducción de oraciones, pero esta a su vez vuelve al programa menos eficiente, por lo cual se recomienda tener un balance entre estas dos consideraciones.
- Se recomienda, al que se interese en el presente proyecto, agregar más gramática al programa, por medio de los hechos en la base de datos. Siguiendo las formas anteriormente descritas, se puede ampliar la capacidad léxica de TransLog.
- Las reglas gramaticales están bastante limitadas, por lo que se recomienda en un futuro agregar más tipos de gramáticas, para poder traducir una mayor cantidad de oraciones.
- Se debe dejar de lado el pensamiento de orientación a objetos o programación procedimental, ya que Prolog trabaja de forma muy distinta, y puede resultar chocante si se trata de entender su funcionamiento de estas formas.

1.8 Bibliografía consultada en todo el proyecto.

Estructura del idioma español:

- <https://espanol.lingolia.com/es/gramatica/estructura-de-la-oracion/oraciones-interrogativas#:~:text=Presentan%20la%20estructura%20siguiente%3A%20interrogativo,como%20sujeto%20de%20la%20oraci%C3%B3n.>
- <https://www.practicaespanol.com/los-adverbios-de-tiempo-lugar-modo-cantidad/>
- <https://educalingo.com/es/dic-es>

Uso de Prolog:

- Concatenar una lista: <https://www.youtube.com/watch?v=eCMRoDsAf0w&t=773s>
- Funcion para concatenar una lista de elementos atómicos:
https://www.swi-prolog.org/pldoc/doc_for?object=atomics_to_string/3
- Libro de Prolog del curso: <http://mural.uv.es/mijuanlo/PracticasPROLOG.pdf>

2. Bitácora.

09 de julio:

- ☐ Se agregan a la base de datos sustantivos, adjetivos, verbos, pronombres, determinantes. Estos se clasifican de acuerdo al número (singular o plural) y de acuerdo al género (masculino, femenino). Saymon y Oscar. 4 horas.

10 de julio:

- ☐ Se investigan los elementos que componen una oración en español. Se plantean ideas para crear la gramática libre de contexto. Saymon. 3 horas.
- ☐ Se investiga sobre la gramática del idioma inglés. Oscar. 2 horas.

11 de julio:

- ☐ Se realizan varias pruebas con la gramática libre de contexto implementada en Prolog. Es una gramática bastante primitiva que solo traduce preguntas, palabras y oraciones demasiado simples. Saymon y Oscar. 3 horas.

13 de julio:

- ☐ Se le agrega a la gramática la posibilidad de reconocer y traducir oraciones que estén relacionadas mediante una conjunción (y/o). Saymon. 3 horas.
- ☐ Se agregan verbos a la base de datos. Oscar. 5 horas.

14 de julio:

- ☐ Se le agrega a la gramática la estructura necesaria para reconocer oraciones que tengan preposiciones (Ej: de, del, con...). Saymon. 2 horas.

15 de julio:

- ☐ Se agregó la gramática necesaria para reconocer signos de interrogación y exclamación en las frases, tanto para inglés, como para español. Pero solo se detecta el signo al final de la oración, el signo al principio se omite. Saymon. 3 horas.
- ☐ Se realizan pruebas generales en el funcionamiento de la gramática libre de contexto. Oscar. 3 horas.

16 de julio:

- ☐ Se le agrega cierta gramática para unos casos de excepción, solo por temas de la especificación de la tarea. Saymon. 1 hora
- ☐ Se comenta el código. Oscar. 1 hora.

17 de julio:

- ☐ Se trabaja en la documentación y en detallar el código fuente. Saymon. 2 horas.
- ☐ Se realiza la documentación del proyecto. Saymon y Oscar. 7 horas.

18 de julio:

- ☐ Se realiza el manual de usuario. Oscar. 3 horas.