Representación semántica de lenguaje natural en el dominio de fórmulas lógicas

Diego Piloni

Directores: Miguel Pagano y Demetrio Vilela

Jueves 5 de Octubre, 2017

FaMAF,
Universidad Nacional de Córdoba

¿De qué voy a hablar?

Problema que quiero resolver

Grammatical Framework

- Herramienta
- Ejemplo simple en GF

GF para resolver el problema de formalización

Conclusión

¿De qué voy a hablar?

Problema que quiero resolver

Grammatical Framework

- Herramienta
- Ejemplo simple en GF

GF para resolver el problema de formalización

Conclusión

Relación entre lógica y lenguaje natural

- La lógica formal deductiva proporciona, según se suele sostener, métodos e instrumentos para el análisis y evaluación de argumentos formulados en lenguaje natural. (General para todas las áreas del conocimiento)
- En el área de la Computación, dado un problema especificado informalmente (usualmente en lenguaje natural), es una tarea importante construir una especificación formal que represente de la manera más fiel posible dicho problema.

Aprendizaje y Enseñanza de formalización

- La enseñanza de formalización, como traducción de un lenguaje no-formal o semi-formal a un lenguaje formal de la lógica, constituye generalmente una parte de los cursos de lógica de nivel universitario.
- (Oller, 2006) Profesores descubren rápidamente que la tarea de formalización resulta difícil, aún más que otras tareas como construcción de demostraciones.

Aprendizaje y Enseñanza de formalización

- La cuestión de las dificultades propias de la formalización no es trivial.
- Se presupone la posibilidad de llevar a cabo exitosamente la traducción de argumentos en lenguaje natural a algún lenguaje de la lógica formal.

Dos fuentes de dificultad cuando se traduce un texto de una lengua de origen a una lengua de destino.

1. Comprensión del texto en la lengua de origen.
2. Producción del texto en la lengua de destino.

Dificultades propias de formalización

Veamos dos ejemplos aparentemente simples y paralelos, en donde se toma como regla de traducción que los sustantivos comunes y adjetivos se traducen a predicados de Lógica de primer orden.

1. ABC es un triangulo equilatero $\rightarrow Tr.ABC \wedge Eq.ABC$
   Parece ser una traducción aceptable.
2. Facundo es un basquetbolista bajo
   $\rightarrow Basq.Facundo \wedge Bajo.Facundo$
   Facundo podría ser considerada una persona alta, pero baja para ser un jugador de basquet.

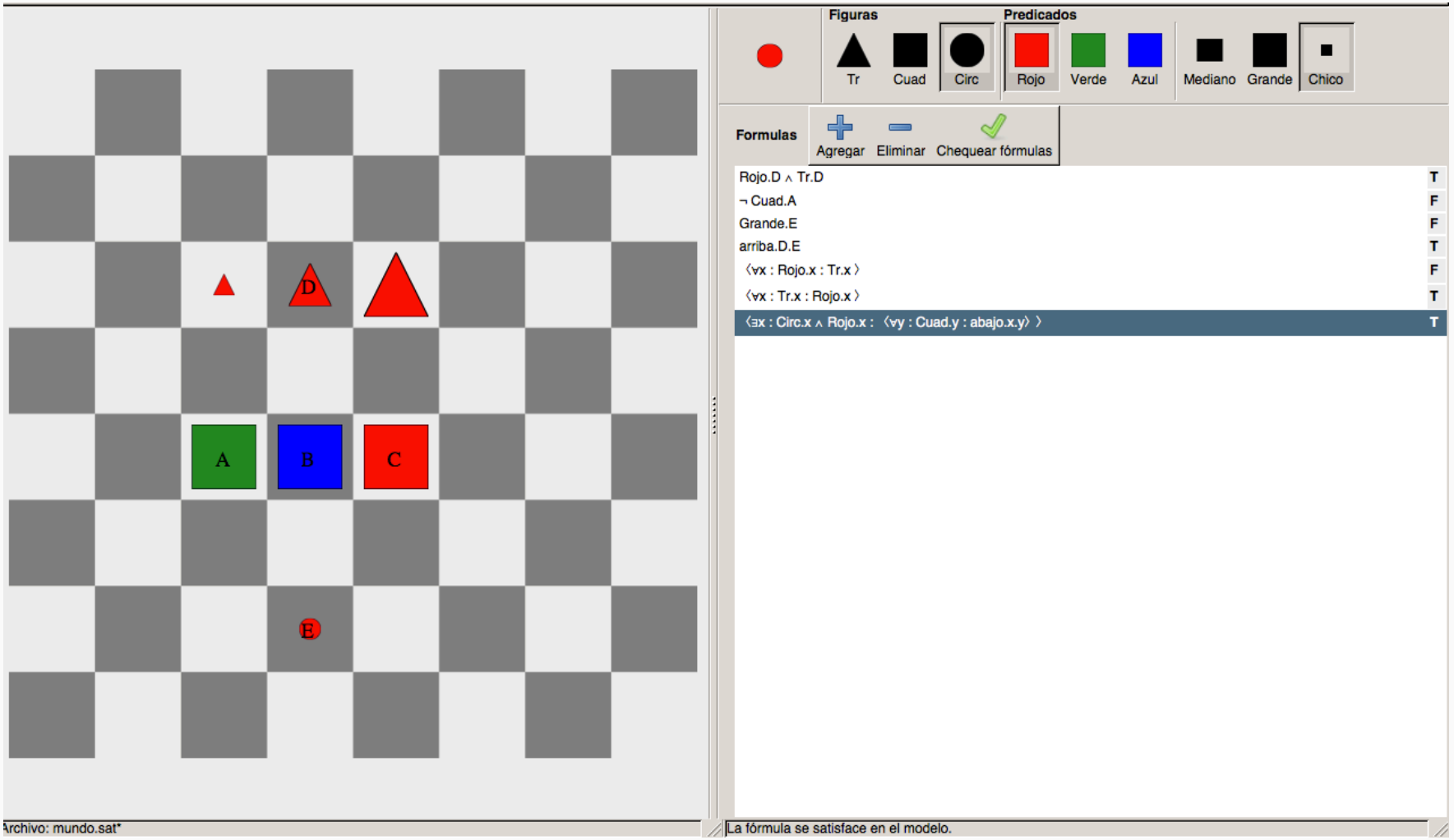
Aprendizaje y Enseñanza de formalización

Si bien existen herramientas didácticas para el aprendizaje de la lógica, la mayoría se centran en:

- Sistemas deductivos
- Semántica formal de fórmulas lógicas (Programas como SAT)

Pareciera, por lo tanto, que faltan herramientas digitales que transparenten las dificultades inherentes a la traducción de lenguajes naturales al lenguaje simbólico y ayuden a realizar esta traducción.

SAT


Figuras
Tr
Cuad
Circ
Predicados
Rojo
Verde
Azul
Mediano
Grande
Chico
Formulas
Agregar
Eliminar
Chequear fórmulas
Rojo.D ∧ Tr.D T
¬ Cuad.A F
Grande.E F
arriba.D.E T
⟨∀x : Rojo.x : Tr.x ⟩ F
⟨∀x : Tr.x : Rojo.x ⟩ T
⟨∃x : Circ.x ∧ Rojo.x : ⟨∀y : Cuad.y : abajo.x.y⟩ ⟩ T
D
A
B
C
E
Archivo: mundo.sat*
La fórmula se satisface en el modelo.

Objetivo de Tesis

El objetivo de la tesis es desarrollar una herramienta que traduzca oraciones del castellano a fórmulas lógicas de primer orden, tomando como referencia SAT.
Ejemplos de traducciones:

- D es grande y cuadrado $\rightarrow Gr.D \wedge Cuad.D$
- A no es cuadrado $\rightarrow \neg Cuad.A$
- A está arriba de D $\rightarrow arr.A.D$
- Todas las figuras rojas son triangulares
  $\rightarrow \langle \forall x : Rojo.x : Tr.x \rangle$
- Existe un círculo rojo que está abajo de todos los cuadrados
  $\rightarrow \langle \exists x : Circ.x \wedge Rojo.x : \langle \forall y : Cuad.y : abajo.x.y \rangle \rangle$

¿De qué voy a hablar?

Problema que quiero resolver

Grammatical Framework

- Herramienta
- Ejemplo simple en GF

GF para resolver el problema de formalización

Conclusión

¿De qué voy a hablar?

Problema que quiero resolver

Grammatical Framework

- Herramienta
  - Ejemplo simple en GF

GF para resolver el problema de formalización

Conclusión

Grammatical Framework

- Grammatical Framework (GF) es un lenguaje de programación open source, diseñado para escribir gramáticas multilingües.
- Creado por: Aarne Ranta, Profesor de Ciencias de la Computación en la Universidad de Gotemburgo.
- webpage: http://www.grammaticalframework.org/

Características de GF

GF es un lenguaje de alto nivel, con las siguientes características:

- Un lenguaje de propósito específico en gramáticas, como YACC, Bison, Happy, BNFC.
- Un lenguaje de programación funcional, como Haskell, Lisp, OCaml.
- Una plataforma de desarrollo para gramáticas de lenguaje natural, como LKB, XLE, Regulus.
- Un framework lógico, como Agda, Coq, equipado con sintaxis concreta en adición a lógica.
- Una plataforma para traducción automática, como Moses, Apertium.

Características de GF

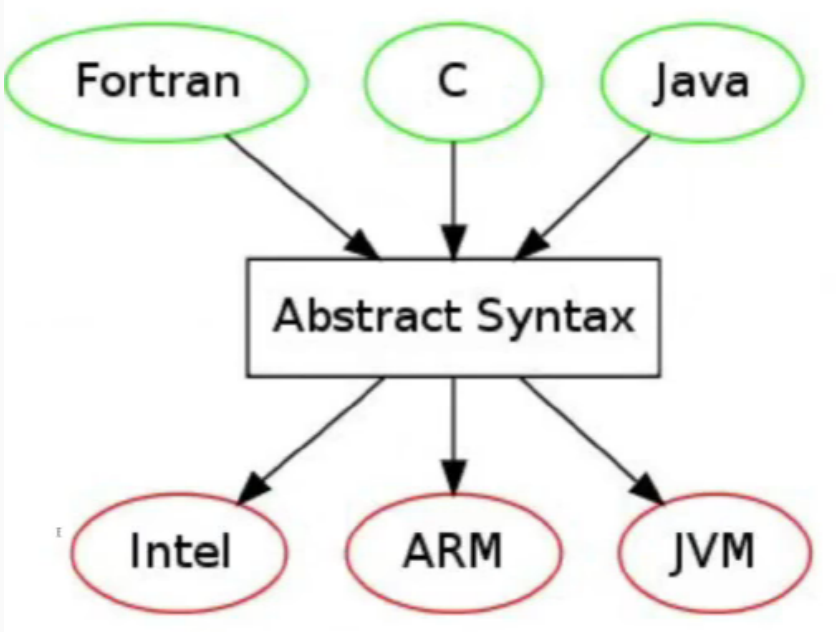
- Las gramáticas se compilan y pueden ser importadas desde otros lenguajes de programación, como Haskell, Python, Java y Javascript.
- GF cuenta con una librería propia llamada Resource Grammar Library (RGL) que provee inflexiones morfológicas y reglas sintácticas.
- La RGL está diseñada para ser usada por expertos de dominio (y no tan expertos, como en mi caso) sin gran entrenamiento lingüístico (¡justo lo que necesito!).

PMCFG

- GF es equivalente en expresividad a PMCFG (Parallel Multiple Context-Free Grammars), que son ligeramente sensibles al contexto y podemos pensarlas como ‘gramáticas sobre tuplas’.
- Teóricamente, el parsing en GF y PMCFG es polinomial.
- En la práctica, gramáticas que usan la RGL suelen ser lineales.
- Dato práctico interesante: GF cuenta con un parser incremental. Muy cómodo para usar en el interprete!

Sintaxis abstracta y sintaxis concreta

El funcionamento de GF está inspirado fuertemente por el funcionamento de compiladores.



Una sintaxis abstracta única, acompañada de una sintaxis concreta para cada lenguaje.

Sintaxis abstracta y sintaxis concreta

Sintaxis abstracta: Mediante una gramática abstracta se definen los posibles árboles de sintaxis abstractos (AST). Estos representan la semántica relevante que es común a todos los lenguajes.

- Categorías (cat): Que definen los tipos de los AST.
- Funciones (fun) sobre las categorías definidas. Estas funciones son las que construyen los AST.

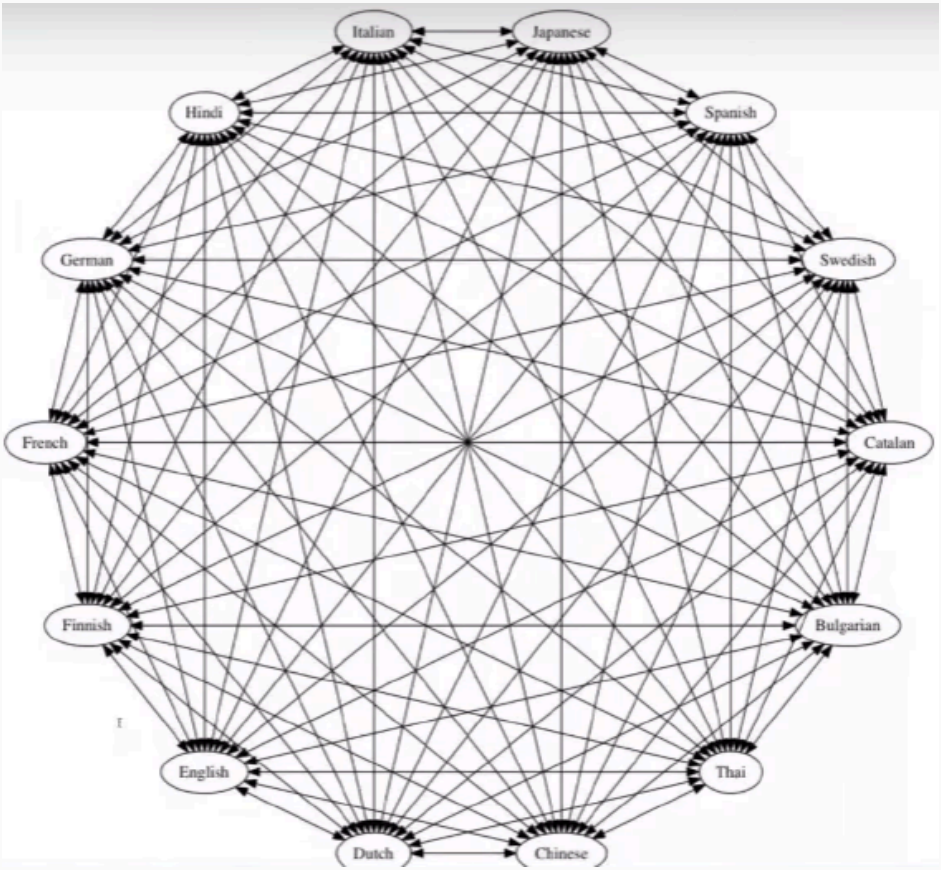
Sintaxis abstracta y sintaxis concreta

Sintaxis concreta: Define la forma en que se linealizan los AST a strings. GF genera automáticamente el parser.

- Categorías de linealización (lincat): Son los tipos concretos que se le asignan a cada categoría de la sintaxis abstracta. Generalmente strings, pero no siempre.
- Reglas de linealización (lin) de los AST, teniendo en cuenta las categorías de linealización.

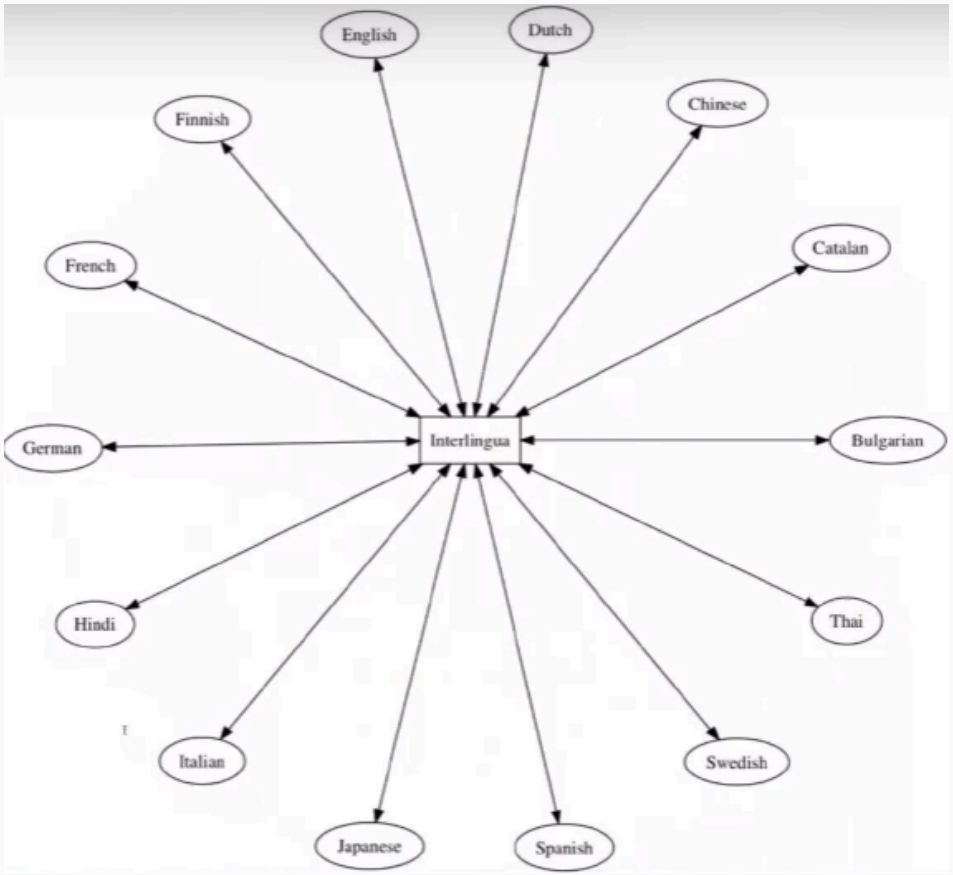
Sintaxis abstracta y concreta en Leng. Natural

Normalmente sería necesario una traducción independiente para cada par de lenguajes.



Sintaxis abstracta y concreta en Leng. Natural

Usando sintaxis abstracta como una especie de ‘interlingua’, podemos definir una traducción entre la ‘interlingua’ y cada lenguaje, y así traducir entre cualquier par de lenguajes.



¿De qué voy a hablar?

Problema que quiero resolver

Grammatical Framework

- Herramienta
- Ejemplo simple en GF

GF para resolver el problema de formalización

Conclusión

Gramática abstracta - Alimentos

```
abstract Food = {
flags startcat = Comment ;
cat
    Comment ; Item ; Kind ; Quality ;
fun
    Pred : Item -> Quality -> Comment ;
    This, That : Kind -> Item ;
    Mod : Quality -> Kind -> Kind ;
    Wine, Cheese, Fish : Kind ;
    Very : Quality -> Quality ;
    Italian, Expensive, Delicious : Quality ;
}
```

Sintaxis concreta - Alimentos Inglés

```
concrete FoodEng of Food = {
lincat
    Comment, Item, Kind, Quality = Str ;
lin
    Pred item quality = item ++ "is" ++ quality ;
    This kind = "this" ++ kind ;
    That kind = "that" ++ kind ;
    Mod quality kind = quality ++ kind ;
    Wine = "wine" ;
    Cheese = "cheese" ;
    Fish = "fish" ;
    Very quality = "very" ++ quality ;
    Italian = "Italian" ;
    Expensive = "expensive" ;
    Delicious = "delicious" ;
}
```

Sintaxis concreta - Alimentos Español

```
concrete FoodSpa of Food = {
lincat
    Comment, Item, Kind, Quality = Str ;
lin
    Pred item quality = item ++ "es" ++ quality ;
    This kind = "este" ++ kind ;
    That kind = "ese" ++ kind ;
    Mod quality kind = quality ++ kind ;
    Wine = "vino" ;
    Cheese = "queso" ;
    Fish = "pescado" ;
    Very quality = "muy" ++ quality ;
    Italian = "italiano" ;
    Expensive = "caro" ;
    Delicious = "delicioso" ;
}
```

Cómo representar variaciones en género y número?

Solución BNF

```
Comment ::= Item_Sg_Masc "es" Quality_Sg_Masc
Comment ::= Item_Sg_Fem "es" Quality_Sg_Fem
Comment ::= Item_Pl_Masc "son" Quality_Pl_Masc
Comment ::= Item_Pl_Fem "son" Quality_Pl_Fem
```

Solución GF, tables

```
param Number = Sg | Pl ;
param Gender = Masc | Fem ;
-- Tipo de adjetivos en español: Gender => Number => Str
-- inflección de italiano
table {
    Masc => table {Sg => "italiano" ; Pl => "italianos"} ;
    Fem => table {Sg => "italiana" ; Pl => "italianas"}
}
```

En la práctica: A las variaciones de género y número las maneja internamente la RGL.

Características variables vs inherentes

- Los adjetivos en español pueden variar tanto en genéro como en número. En GF se usan tables para representar esta variación.
- Los sustantivos comunes varían en número, pero tienen género fijo. Por ejemplo: Pizza tiene género femenino, pero puede variar en número: Pizza o Pizzas.

Para representar características inherentes, GF provee records.

```
-- Records en GF
-- tipo : { s : Num => Str ; g : Gender }
{s = table {Sg => "pizza" ; Pl => "pizzas"} ; g = Fem}
```

Otras características interesantes de GF

Como Lenguaje de programación:

- Herencia de módulos,
- Aplicación parcial,
- Funciones polimórficas, y
- Pattern matching, sobre ADTs y sobre strings

Sintaxis abstracta:

- Funciones de alto orden, y
- Tipos dependientes.

Sintaxis concreta:

- Definición de operaciones: útiles para no repetir código.
- Resource modules pueden ser usados desde múltiples módulos concretos.
- Funtores (Módulos paramétricos) lenguajes comparten cierta estructura.

¿De qué voy a hablar?

Problema que quiero resolver

Grammatical Framework

- Herramienta
- Ejemplo simple en GF

GF para resolver el problema de formalización

Conclusión

GF para resolver el problema de formalización

Categorías de gramática abstracta base

Categoría	Descripción	Ejemplo
Prop	proposición, compleja o atómica	A es rojo y B es azul
Atom	propisición atómica	A es rojo
Pred1	Predicado unario	rojo
Pred2	Predicado binario	arriba de
Ind	término indivual	A
Var	variable de cuantificación	x
Fun1	función unaria	$\_^2$
Fun2	función binaria	$\_ + \_$
Conj	conjunción	y, o
Quant	símbolo de cuantificación	$\forall$, $\exists$
Kind	dominio de cuantificación	figura

Prop

Prop es la categoría usada para representar cualquier proposición de primer orden válida. Con los siguientes constructores:

```
True : Prop
False : Prop

PAtom : Atom -> Prop
-- A está arriba de B

PNegAtom : Atom -> Prop
-- A no es rojo

PNeg : Prop -> Prop
-- No es (cierto | el caso) que ..

PConj : Conj -> Prop -> Prop -> Prop
-- A es rojo y B es grande
```

Prop

```
PImpl : Prop -> Prop -> Prop
-- si todas las figuras son rojas entonces A es rojo

PQuant : Quant -> Var -> Prop -> Prop -> Prop
-- Cuantificación de la forma <Qx : R.x : T.x>

UnivIS : Var -> Kind -> Pred1 -> Prop
-- todas las figuras rojas son grandes

ExistIS : Var -> Kind -> Pred1 -> Prop
-- alguna figura chica está arriba de A
```

Atom

Atom es la categoría usada para representar proposiciones atómicas. Con los siguientes constructores:

```
APred1 : Pred1 -> Ind -> Atom
APred2 : Pred2 -> Ind -> Ind -> Atom
```

Nota: Pred1 y Pred2 son parte del lexicón.

Kind

Kind es la categoría utilizada para definir los dominios de cuantificación in-situ. Es más cercano al lenguaje natural que al lenguaje simbólico.

```
Figura : Kind
ModKind : Kind -> Pred1 -> Kind
```

Nota: Kind es parte del lexicón.

Cosas que quedaban fuera de esa gramática y fui agregando

- Conjunción de individuos
  - Julia y María son mujeres, en vez de, Julia es mujer y María es mujer
  - A, B y C son rojos, en vez de, A es rojo, B es rojo y C es rojo
- Conjunción de predicados
  - Juan es alto y flaco, en vez de, Juan es alto y Juan es flaco
  - A es rojo y grande, en vez de, A es rojo y A es grande
- Listas de proposiciones
  - P, Q y R, en vez de, P y Q y R

Cosas que quedan afuera de esa gramática y fui agregando

- Distribución de predicados binarios (No todos, se usa un tipo dependiente para seleccionar)
  - A, B y C son amigos, en vez de, A es amigo de B, B es amigo de C y A es amigo de C
  - A, B y C son iguales, en vez de, A es igual a B , B es igual C y A es igual a C
- Aplicación parcial de Pred2 en Pred1 (Principalmente útil para usar Pred2 en cuantificación)
  - todas las figuras que están arriba de A son grandes

Funciones de transferencia

- Al extender la gramática con los puntos anteriores hay AST's cuya linealización al lenguaje simbólico de la lógica no es directa.
- Es necesario definir *funciones de transferencia*, para transformar los AST's generados al parsear del español a otros AST's, cuya linealización sea directa.
- Definir correctamente funciones de transferencia para cada caso que se genera es lo que más trabajo lleva.

Pipeline con funciones de transferencia

Luego, el pipeline de traducción es el siguiente:

1. Parsear frase en lenguaje natural, esto genera un conjunto de árboles AST_0
2. Aplicar función de transferencia correspondiente a cada árbol de AST_0, lo que nos devuelve un nuevo conjunto AST_1
3. Linealizar cada árbol de AST_1, lo que generará una fórmula de primer órden por cada árbol.

Aplicación de predicado a conjunción de individuos

`input`$_0$`: `A, B y C son rojos'

ast_0`: PAtom (APred1 Rojo (ConjInd CAnd [A, B, C]))`

ast_1`: PConj CAnd (PAtom (APred1 Rojo A))`
`(PConj CAnd (PAtom (APred1 Rojo B))`
`(PAtom (APred1 Rojo C)))`

`output`$_0$`:` $Rojo.A \wedge Rojo.B \wedge Rojo.C$

Aplicación de predicado a conjunción de individuos

`input`$_1$`: ` `` `A y B estan arriba de C' ``

ast_0`: PAtom (APred2 Arriba (ConjInd CAnd [A, B]) C)`

ast_1`: PConj CAnd (PAtom (APred2 Arriba A C))`
`(PAtom (APred2 Arriba B C))`

`output`$_1$`:` $arriba.A.C \wedge arriba.B.C$

¿De qué voy a hablar?

Conclusión

Ventajas de trabajar con GF en este problema:

- La **precisión** de la traducción está programada.
- Al traducir a un lenguaje formal es importante que la traducción sea correcta.
- Se puede garantizar que la traducción sea correcta.
- La RGL facilita la definición de la sintaxis del español.

Desventajas:

- 'All grammars leak'.
- Completar la gramática para hacerla cada vez más natural o **robusta** lleva su tiempo.
- Necesidad de conocimiento de area.
- ¿Cómo testear?