Intérprete de reglas para la obtención de analizadores-generadores eficientes del lenguaje natural

Antonio Menchén Peñuela

Dpto. de Lenguajes y Sistemas Informáticos
Universidad de Sevilla
Facultad de Informática

E-mail: atenea.lsi.us.es

Resumen

En este trabajo se propone un intérprete de reglas que permite obtener programas lógicos que no presentan ciertas ineficiencias. Estos programa pueden usarse a la vez como analizadores y generadores del lenguaje natural.

Palabras clave: programación lógica, intérprete de reglas, formas lógicas, gramáticas basadas en la unificación o la lambda-reducción de formas lógicas, intérprete de reglas, procesamiento del lenguaje natural.

Gramáticas de unificación

Uno de los formalismos gramaticales más indicados para el procesamiento del lenguaje natural (PLN), es el que se engloba bajo el nombre de Gramáticas de Unificación (UGs).

Una regla UG tiene el aspecto general:

$X_0 \rightarrow X_1 \ldots X_N$

$FS_0 = FS_I = \ldots = FS_J$

...

$FS_M = FS_K = \ldots = FS_L$

con $N >= 1$, $M < N$, el resto de índices están comprendidos entre 1 y N, y las FSs son rasgos asociados a los símbolos gramaticales.

Debido a que el rasgo **categoría gramatical**, está siempre presente, se le asigna un papel relevante y las reglas UGs se suelen escribir como:

$C_0 \rightarrow C_1 \ldots C_N$

$FS_0 = Fs_I = \ldots = FS_J$

...

$FS_M = FS_K = \ldots = FS_L$

Uno de los aspectos que hacen interesante este formalismo, es que los analizadores obtenidos a partir de él son a su vez generadores, o, si nos centramos en el rasgo semántico o forma lógica, LF, diremos que el intérprete semántico es también el realizador semántico.

Sin embargo, las UGs presentan entre otras las siguientes desventajas:

1. Son muy complicadas debido al gran número de rasgos que es necesario tener en cuenta para ciertos símbolos.

2. Hay aspectos del PLN que no se pueden tratar con este formalismo (en [Allen,94] se discute el caso de las sentencias con sujetos coordinados).

3. El intérprete semántico usado como realizador semántico presenta ineficiencias.

La primera desventaja tiene su origen en el propio diseño de una regla UG: se maneja el concepto de **símbolo-cabeza**, en el sentido de que su rasgo semántico es el del símbolo de la izquierda, y en el resto del conjunto de rasgos asociados a este símbolo están los rasgos semánticos asociados a los demás símbolos de la derecha. Por lo tanto, en las UGs los símbolos-cabeza presentan una gran complicación.

La razón de la tercera desventaja apuntada es porque a menos que el símbolo-cabeza esté el primero entre los símbolos de la derecha (cosa que no ocurre con frecuencia), cada símbolo de la derecha anterior a él generará todas las sentencias posibles, ya que en la realización semántica se conoce el rasgo semántico del símbolo de la izquierda pero no el de cada símbolo de la derecha (exceptuando el del símbolo-cabeza). Esto hará que el intérprete semántico vuelva atrás repetidas veces reescribiendo todas las reglas que tienen como símbolo de la izquierda a los

símbolos apuntados, hasta que generen la parte de la sentencia anterior a la que se corresponde con el rasgo semántico del símbolo-cabeza.

Un ejemplo aclarará lo dicho. Supongamos la regla UG para la sentencia completa en inglés:

S -> NP VP
$SEM_S = SEM_{VP}$
$SUBJ_{VP} = SEM_{NP}$

Si se conoce **SEM_S**, hasta que no se reescriba el **vp** (que es el símbolo-cabeza), **np** ligará **SEM_{NP}** a todos los valores posibles, lo cual dará lugar a todas las partes de la sentencia (predicados nominales) posibles anteriores a la asociada al símbolo-cabeza. Esto puede ser muy ineficiente si pensamos en gramáticas extensas.

En lo que sigue, presentaré primeramente otro formalismo, también muy usual en el PLN, y estudiaré sus aspectos generales, para pasar posteriormente a proponer un tercer formalismo que admita las ventajas de los dos primeros, y no tenga las desventajas apuntadas. Esto se conseguirá, finalmente, mediante la aplicación de un intérprete de reglas que será presentado y discutido. En un último apartado del trabajo, discutiré las conclusiones y trabajos futuros relacionados con el presente tema.

El cálculo lambda en las gramáticas para el PLN

El cálculo lambda es un formalismo matemático que en el PLN ha sido utilizado para establecer las ligaduras entre los rasgos semánticos asociados a los símbolos gramaticales. En esencia consiste en asociarle al símbolo-cabeza un rasgo semántico que es una función (lambda-expresión). El número de parámetros formales de esta lambda-expresión, se corresponde con el número de símbolos de la derecha, y cada parámetro real con cada rasgo semántico de estos símbolos. El rasgo semántico del símbolo de la izquierda es la lambda-reducción.

De esta forma, el ejemplo anterior de regla UG se escribiría ahora como:

S -> NP VP

$SEM_S = (SEM_{VP}\ SEM_{NP})$

La ventaja de utilizar este formalismo es que los conjuntos de ras-

gos asociados a los símbolos-cabeza son mucho menos complicados que los vistos para las UGs, (obsérvese que ha desaparecido el rasgo $\mathbf{SUBJ_{VP}}$), y se pueden tratar aspectos del PLN que no trataban las UGs.

La desventaja más clara es que de estas reglas se obtienen realizadores semánticos ambiguos.

Esto puede verse tomando la gramática anterior y haciendo $\mathbf{SEM_S}$ = **(A B C)**.

Esta lambda-reducción puede obtenerse, por ejemplo, de dos formas:

(lambda X.(A X C) B)

y

(lambda X. (A B X) C)

Por lo tanto, $\mathbf{SEM_{VP}}$ puede adoptar 2 formas distintas que se corresponden con sentencias diferentes.

Formalismo alternativo

Parece pues necesario adoptar un formalismo que reúna las ventajas de los dos anteriores, y reduzca las desventajas de los mismos.

Evidentemente, el 2° formalismo comentado se debe rechazar de pleno, puesto que da lugar a intérpretes semánticos ambiguos. La metodología que se propone aquí es convertir cualquier gramática escrita en los dos formalismos comentados, en una gramática escrita en el formalismo que va a desarrollarse a continuación, de una manera totalmente **transparente** al diseñador de gramáticas.

Una alternativa aceptable es proponer un formalismo que esté basado en la unificación de rasgos, donde queden eliminadas las ineficiencias en la realización semántica, y que pueda originarse por la traducción de gramáticas basadas en el cálculo lambda de LFs.

Veamos un ejemplo:

La regla:

S -> NP VP

$SEM_S = (SEM_{VP}\ SEM_{NP})$

puede traducirse a:

S -> NP VP
$SEM_S = SEM_{VP}$
$SUBJ_{VP} = SEM_{NP}$

y ésta a su vez a las dos reglas:

S -> NP VP

{(NONVAR $STRING_S$)
$SEM_S = SEM_{VP}$
$SUBJ_{VP} = SEM_{NP}$}

S -> VP NP

{(NONVAR SEM_S)
$SEM_S = SEM_{VP}$
$SUBJ_{VP} = SEM_{NP}$}

Como puede observarse el formalismo que propongo es muy parecido al UG pero las ecuaciones de restricciones están guardadas. El significado de estos guardas es que sólo se aplican estas ecuaciones si el guarda se cumple.

También se observará que he tomado como rasgo asociado a un símbolo gramatical una parte de la sentencia. Esto permite formalizar lo que comentábamos de las UGs al afirmar que el intérprete semántico es también el realizador semántico.

Si se cumple el primer guarda estaremos en el caso de la interpretación semántica más usual (el análisis), y si se cumple el segundo guarda obtendremos la realización semántica (generación). Un detalle importante es que en este último caso el símbolo-cabeza se ha colocado al principio de los símbolos de la parte derecha.

El hecho de que se dupliquen las reglas en las que hay ligaduras en el rasgo semántico, puede añadir ineficiencias que antes no existían en el intérprete semántico. Sin embargo, como se discutirá más adelante, dado que va a desarrollarse el formalismo dentro de la programación lógica, es fácil eliminar estas ineficiencias añadiendo en el cuerpo de la primera regla un corte.

Por darle un nombre con el que distinguir el formalismo propuesto, lo voy a llamar Gramáticas Alternativas de Unificación (UAGs).

Las UAGs utilizando programación lógica

Las reglas:

S -> NP VP

{(NONVAR $STRING_S$)
SEM_S = SEM_{VP}
$SUBJ_{VP}$ = SEM_{NP}}

S -> VP NP

{(NONVAR SEM_S)
SEM_S = SEM_{VP}
$SUBJ_{VP}$ = SEM_{NP}}

puestas en la forma conveniente, se traducen al aplicarle el compilador de Gramáticas de Cláusulas Definidas (DCGs), en las cláusulas:

(1)

```
s(SEM, S, SR) <-
 novar([S,SR] ) & ! &
 np(SUBJ, S, SR1) &
  vp(SEM,SUBJ, SR1, SR).

s(SEM, S, SR) <-
 novar(SEM) &
 vp(SEM,SUBJ, SR1, SR) &
  np(SUBJ, S, SR1).
```

Si tomamos de nuevo la regla para la sentencia utilizando el formalismo de lambda reducción de LFs, tendremos que:

S -> NP VP
SEM_S = (SEM_{VP} SEM_{NP})

Puesta en forma adecuada para el compilador DCG, daría lugar a la cláusula:

```
s(SEM_S,S,SR) ->
 np(SEM_NP,S,SR1) &
 vp(SEM_VP,SR1,SR) &
   reduce(SEM_VP, SEM_NP,
SEM_S ).
```

El intérprete que voy a proponer traduce cláusulas como esta última a cláusulas del tipo **(1).**

Antes de seguir, he de apuntar que el método presentado sólo es aplicable a intérpretes semánticos que obtienen LFs donde no se establecen los ámbitos de los cuantificadores (están sin desambiguar). Esto simplifica bastante las gramáticas.

Un intérprete de reglas para la obtención de analizadores-generadores del lenguaje natural eficientes

Para construir el intérprete de reglas que buscamos, pensemos en que vamos a encontrarnos con dos clases de reglas: Las que tienen un símbolo-cabeza, y las que no. Dentro de las primeras también tendremos que tener en cuenta si el símbolo de la izquierda es cabeza de otra regla o no, ya que en el primer supuesto, ha añadirse un parámetro más (en el ejemplo visto este parámetro es **SUBJ**).

Con todo ello se propone el siguiente intérprete:

```
interpreta(Regla),
clause(Regla,Cuerpo),
Regla =..[P,SEM,D1,D2],
  busca_cabeza( Cuerpo, Cabeza, Reduce, Resto1, Resto2, []),
  Cabeza =.. [C,SEM_C,S,SR],
  (Regla1 =.. [P, SEM, X1, D1, D2],
clause(cabeza(Regla1),true),
  (Reduce =..
[R1,SEM_C,SEM_P1,SEM] ->
    Cabeza1 =..
[C,SEM,X1,SEM_P1,S,SR];
    Reduce =.. [R1, SEM_C, SEM_P1,
SEM_P2, SEM] ->
     Cabeza1 =.. [C, SEM, X1,
SEM_P1, SEM_P2, S, SR]);


  Regla1 =.. [P, SEM, X1, X2, D1, D2],
clause(cabeza(Regla1),true),
  (Reduce =..
[R1,SEM_C,SEM_P1,SEM] ->
    Cabeza1 =..
[C,SEM,X1,X2,SEM_P1,S,SR];
    Reduce =.. [R1, SEM_C, SEM_P1,
SEM_P2, SEM] ->
     Cabeza1 =.. [C, SEM, X1, X2,
SEM_P1, SEM_P2, S, SR]);


  Regla1 = Regla,
  (Reduce =..
[R1,SEM_C,SEM_P1,SEM] ->
    Cabeza1 =..
[C,SEM,SEM_P1,S,SR];
    Reduce =.. [R1, SEM_C, SEM_P1,
SEM_P2, SEM] ->
     Cabeza1 =.. [C, SEM, SEM_P1,
SEM_P2, S, SR])),

     assert(cabeza(Cabeza1)),

  retract((Regla:-Cuerpo)),
  assert(( Regla1 :- nonvar([D1, D2]),
!,    Resto1, Cabeza1, Resto2)),
  assert((Regla1 :- nonvar(SEM), !,
Cabeza1, Resto1, Resto2)),!.


interpreta(Regla):-
clause(Regla,Cuerpo),
Regla =.. [P,X^Y^LF,D1,D2],
  Regla1 =.. [P,LF,Y,X,D1,D2],
  retract((Regla:-Cuerpo)),
  assert((Regla1:-Cuerpo)).
```

donde la únicas lambda-expresiones que consideramos tienen dos parámetros, y vienen dadas por los términos de la forma **X^Y^LF**.

Conclusiones y trabajos futuros

Aunque el método presentado pueda parecer bastante particular (se ha propuesto el intérprete pensando en literales con tres argumentos y en un orden determinado), esto puede generalizarse a un número cualquiera de argumentos. Ahora bien, para poder construir el intérprete es necesario que partamos de compiladores más indicados para el PLN que el compilador para DCGs. Actualmente trabajo en un compilador de reglas que utiliza grafos dirigidos acíclicos (DAGs), para representar los conjuntos de rasgos asociados a los símbolos de la gramática, de manera que un conjunto de rasgos se comporta respecto a la unificación como DAG. Me refiero al compilador GULP (Graphs Unification Language Programming), [Covington,89], que es una extensión de PROLOG debido a investigadores de la universidad americana de Georgia. La ventaja de usar este compilador es que podemos olvidarnos del número y orden de los parámetros, ya que todos están presentes en **un** solo argumento, por lo que el caso general se reduciría al tratado con tres argumentos.

Por último diré que el intérprete que se presenta en este trabajo junto con el propuesto en [Menchen,96], han sido probados con éxito a partir de gramáticas DCGs, y, como se insiste allí, aunque los programas que se obtienen resulten más complicados, se cubre perfectamente el objetivo marcado, que no es otro que el diseñador de gramáticas disponga de una herramienta que le permita escribir sus gramáticas con toda naturalidad, sin preocuparse por cuestiones de eficiencia, quedando éstas resueltas en los programas compilados e interpretados que subyacen.

Bilbiografía

[Allen,94] **James Allen.** *Natural language understanding.* Second Edition. **The Benjamin/Cummings Publishing Company.** 1994

[Menchen,96] **Antonio Menchén.** *Intérprete de reglas para aumentar la eficiencia de un programa lógico.* **AEPIA Nº** 71996

[Covington,89] **M. Covington.** GULP *2.0:A extension of PROLOG for unification - Based Grammars.* **Report University of Georgia** 1989