MorphTrans: un lenguaje y un compilador para especificar y generar módulos de transferencia morfológica para sistemas de traducción automática

Alicia Garrido-Alenda y **Mikel L. Forcada**
interNOSTRUM
Departament de Llenguatges i Sistemes Informàtics
Universitat d'Alacant
E-03071 Alacant, Spain.
http://www.internostrum.com

Resumen Este artículo presenta un lenguaje para especificar las reglas de un módulo de transferencia morfólogica para un sistema de traducción automática (TA) así como el compilador que se tiene que utilizar para convertir esta especificación en un programa ejecutable. El módulo de transferencia morfológica trabaja sobre la salida que produce un analizador morfológico seguido de un desambiguador léxico categorial, es decir, trabaja sobre secuencias de formas léxicas (lema, categoría léxica y características morfológicas). Un conjunto de reglas especifica un módulo que detecta la secuencia de formas léxicas más larga entre aquellos patrones especificados, consulta el diccionario bilingüe para todas las formas léxicas, las manipula y a continuación escribe la secuencia de formas léxicas correspondiente en la lengua meta. El lenguaje está diseñado de manera que un lingüista pueda escribir reglas fácilmente; a partir de ellas el compilador genera un programa `lex` que realiza la tarea. Este sistema ha sido utilizado para construir el módulo de transferencia de interNOSTRUM, un sistema de TA castellano–catalán.

1 Introducción

La mayoría de sistemas de TA están organizados en torno a la denominada *arquitectura de transferencia* (Arnold et al. 1994; Arnold 1993; Hutchins y Somers 1992): el texto en la lengua origen (LO) es analizado y transformado en una representación abstracta (*análisis*) que es convertida (*transferencia*) en una representación similar pero en la lengua meta (LM), y finalmente, el texto de la LM es generado a partir de esta última representación (*generación*). El sistema de TA castellano–catalán interNOSTRUM (descrito en este mismo congreso (Canals-Marote et al. 2001) y disponible en `http://www.internostrum.com/`) utiliza una estrategia de transferencia morfológica avanzada que es muy parecida a la utilizada en sistemas de TA comerciales para PC tales como Transcend RT de Transparent Technologies, las primeras versiones de Power Translator de Globalink y Reverso de Softissimo. El funcionamiento de interNOSTRUM tiene las siguientes fases:

1. ANÁLISIS
   - Análisis morfológico
   - Desambiguación léxica categorial (estadística)
2. TRANSFERENCIA
   - Consulta al diccionario bilingüe
   - Procesamiento de patrones de palabras (concordancias, reordenación, cambios léxicos)
3. GENERACIÓN
   - Generación morfológica
   - Postgeneración (reglas para la apostrofación y guionado en catalán).

En particular, el diseño del módulo de transferencia en interNOSTRUM está basado en un cuidadoso estudio de "caja negra" de las estrategias de transferencia de los sistemas de TA mencionados anteriormente (para detalles, consúltese Mira i Gimènez y Forcada (1998); Forcada (2000)).

2 El módulo de transferencia

La tarea de transferencia está organizada en *patrones* que representan secuencias de formas léxicas de la lengua origen (FLLO) de longitud fija; una secuencia sigue un cierto patrón cuando contiene la secuencia de categorias léxicas correspondiente. El sistema

contiene un catálogo de patrones que sabe como procesar. Los patrones no son "frases" ni constituyentes en un sentido sintáctico estricto, porque son planos y sin estructura, pero la detección de patrones es un avance con respecto al mero análisis morfológico y puede ser considerado como una forma rudimentaria de análisis sintáctico, de ahí el nombre de *transferencia morfológica avanzada*.

La *detección de patrones* funciona como sigue: si el módulo de transferencia empieza a procesar la *i*ésima FLLO del texto, l_i, éste intenta comparar la secuencia de FLLO $l_i, l_{i+1}, \ldots$ con todos los patrones que hay en su catálogo: escoge el patrón más largo que coincida, procesa la secuencia detectada (mirar más abajo), y el proceso continúa en la FLLO l_{i+k}, donde k es la longitud del patrón que se acaba de procesar[1]. Si ningún patrón coincide con la secuencia que empieza con la FLLO l_i, ésta se traduce como una palabra aislada y el proceso comienza de nuevo con la FLLO l_{i+1} (es decir, cuando ningún patrón es aplicable, el sistema recurre a la traducción palabra por palabra). Nótese que cada FLLO es procesada una sola vez: los patrones no se solapan; por tanto, el procesamiento se realiza de izquierda a derecha y en fragmentos bien definidos.

El procesamiento de patrones toma la secuencia de FLLO detectada y construye (utilizando un programa para la consulta del diccionario bilingüe) una secuencia de formas léxicas en la lengua meta (FLLM) que puede estar reordenada, y de la que se pueden haber eliminado o a la que se pueden haber añadido FL. La información sobre la flexión de las FLLM es generada de manera que se observe la concordancia dentro de la secuencia (si es necesario). Por ejemplo, el patrón castellano artículo–adjetivo–nombre (como puede ser "una señal inequívoca") se convierte en una secuencia sin reordenar en catalán ("un senyal inequívoc"), después de propagar el género masculino en catalán (era femenino en castellano) de "senyal" al artículo y al adjetivo.[2]

[1] En los casos en los que pueda existir ambigüedad debida a la presencia de más de un patrón que coincida con la entrada, la elección del patrón más largo la resuelve de manera expeditiva.

[2] Además, el módulo de transferencia puede mantener información de "estado" para asegurar la relación entre patrones como la concordancia de número entre sujeto y verbo. Esta información de estado se actualiza después del procesamiento de cada patrón.

Naturalmente, un catálogo finito de secuencias "congeladas" de longitud fija no puede cubrir todas las posibles formas que un cierto constituyente (p.e. un sintagma nominal) pueda tomar, pero si los patrones son escogidos de manera que cubran los fenómenos más comunes, se puede obtener una calidad razonable, en particular cuando, como es el caso de interNOSTRUM, las lenguas origen y meta son sintácticamente similares. Este diseño —que favorece los patrones más largos— permite a los desarrolladores construir primero un sistema palabra por palabra (que es como funciona por defecto el módulo de transferencia salvo que un patrón sea detectado) y añadir patrones incrementalmente con la confianza de que, si los patrones y sus acciones asociadas están definidos correctamente solo se pueden producir mejoras en la calidad de la traducción.

Tenemos que resaltar que una de las opciones en el diseño de interNOSTRUM es que todos los módulos se comunican con los demás a través de flujo de texto por dos razones: primero, para un diagnóstico de errores fácil durante el desarrollo del sistema y segundo porque es una elección natural cuando se utiliza un sistema operativo orientado a texto como Linux (o, más generalmente, Unix); esto causa una pequeña cantidad de reanálisis a la entrada de cada módulo. Aún así, la velocidad obtenida actualmente en interNOSTRUM sobrepasa las mil palabras por segundo en un PC típico de 1998. El módulo de transferencia, por tanto, lee FLLO de la entrada estándar y escribe FLLM por la salida estándar.

3 Especificación del módulo de transferencia

En lugar de programar el módulo de transferencia directamente en C o en un lenguaje similar, hemos diseñado un lenguaje de alto nivel (con palabras clave en catalán) que puede ser usado por un lingüista, después de un período de entrenamiento[3], para codificar:

- los patrones de FLLO que tienen que ser detectados para su procesamiento;
- la extracción de características gramaticales como el lema, género o número de las FLLO;

[3] del orden de un mes en el largo la resuelve de manera expeditivaproyecto interNOSTRUM

- la manipulación de FLLO y el montaje de sus traducciones (consultadas en un diccionario bilingüe), junto con otras FLLM generadas internamente (si es necesario) para producir el patrón meta.

3.1 Estructura del programa

El programa tiene dos secciones: la sección de declaraciones y la sección de reglas.

La sección de declaraciones: Esta sección incluye:

- La palabra clave `separa` seguida de una expresión regular que define los espacios en blanco que pueden ser encontrados entre formas léxicas. En interNOSTRUM un módulo anterior al de análisis encapsula los blancos y material de formato de RTF y HTML entre dobles corchetes para que sea tratado como espacios en blanco simples en todos los módulos. Este material será usado para reconstruir el formato después de la generación.
- La palabra clave `lema` seguida de una expresión regular define la parte de la forma léxica que contiene el lema (se asume que el lema es la primera parte de una forma léxica). Por ejemplo, "`[A-Za-z]+`" (en interNOSTRUM esta expresión es demasiado larga como para reproducirla aquí ya que contiene caracteres acentuados y signos de puntuación, que son también tratados como palabras).
- La palabra clave `formlex` seguida de una expresión regular define la forma léxica genérica. En interNOSTRUM esta forma léxica genérica consiste en la expresión regular del lema seguida de "`[^/]+"/"`" (cualquier cosa que empiece con un lema y acabe con "/" es una forma léxica).
- La palabra clave `cua` seguida de una expresión regular define la parte de la forma léxica que viene a continuación del lema y la categoría. En interNOSTRUM se trata de "/".
- Una serie de *declaraciones de categorías léxicas* consistentes en la palabra clave `catlex`, un identificador, el símbolo `:=` y una expresión regular que seleccionará aquellas formas léxicas que serán tratadas como una categoría particular. Hemos de destacar que el lingüista puede incluir cualquier información de la forma léxica para definir una categoría; las categorías pueden ser muy genéricas (p.e. todos los nombres) o muy específicas (p.e. sólo aquellos determinantes que son demostrativos plurales).
- Una serie de declaraciones de *atributos*, que consisten en la palabra clave `atribut`, un identificador, el símbolo `:=` y una expresión regular que describe los posibles valores que se pueden encontrar en una forma léxica para un cierto atributo (como pueden ser el *género* o el *número*).
- Una serie de declaraciones de *variables de estado*, que consiste en la palabra clave `estat` y un identificador. Las variables de estado se usan para transferir valores de atributos activos (mirar la palabra clave `activa` más abajo) de un patrón a otro.

La sección de reglas: Esta sección es una secuencia de reglas (en formato libre) patrón-acción. Cada regla tiene tres partes:

- La definición del patrón que será detectado, consistente en la palabra clave `detecta` y una secuencia de categorías léxicas previamente declaradas con `catlex`. Cabe resaltar que, si la entrada coincide con dos reglas diferentes, primero, prevalece la más larga y, segundo, para patrones de la misma longitud, prevalece la regla definida en primer lugar. La longitud de los patrones sólo está acotada por la capacidad de memoria de el sistema.
- Una declaración de aquellos atributos que son relevantes para una regla particular y que por tanto han de ser activados, es decir, extraídos de las FL. Los nombres de atributos siguen a la palabra clave `activa`. Además de los atributos definidos con `atribut`, las categorías léxicas definidas con `catlex` también son extraídas de las FL y pueden ser manipuladas.
- Finalmente, la acción a realizar para el patrón, encapsulada entre llaves y escrita utilizando las reglas del siguiente apartado.

3.2 El lenguaje de las acciones

Las acciones son secuencias de sentencias especificadas en un lenguaje de alto nivel, que

recuerda a C, y que puede ser especificado por la siguiente gramática:

Stats	→	Stat `;` Stats
Stats	→	Stat
Stat	→	{ Stats }
Stat	→	`si (` Cond `)` Stat
Stat	→	`si (` Cond `)` Stat `altrament` Stat
Stat	→	`envia` Expr
Stat	→	`$` **num** `.orig.` **id** `:=` Expr
Stat	→	`$` **num** `.meta.` **id** `:=` Expr
Stat	→	**id** `:=` Expr
Stat	→	
Cond	→	Cond `o` Conj
Cond	→	Conj
Conj	→	Conj `i` SimCon
Conj	→	SimCon
SimCon	→	`(` Cond `)`
SimCon	→	`no` SimCon
SimCon	→	Expr **compop** Expr
Expr	→	Expr SimplExp
Expr	→	SimplExp
SimplExp	→	`net` SimplExp
SimplExp	→	`$` **num** `.orig.` **id**
SimplExp	→	`$` **num** `.meta.` **id**
SimplExp	→	`&` **num**
SimplExp	→	**string**
SimplExp	→	**id**

donde: `si (...) ...altrament ...` es la sentencia condicional; `$` seguido de un número positivo i (**num**) representa la $i$ésima FLLO en el patrón actual; el símbolo `&` seguido de un número positivo i (**num**) representa el espacio en blanco entre la $i$ésima y la $i$ésima+1 forma léxica; el operador `:=` asigna el valor de la parte derecha a la parte izquierda; `envia` manda una expresión a la salida estándar; **compop** representa uno de los operadores de comparación de cadenas `==` (igual) o `!=` (diferente); los operadores `no`, `i` y `o` son los operadores lógicos *not*, *and* y *or* respectivamente; **string** es una cadena entrecomillada, y `.orig.` y `.meta.` son utilizadas respectivamente para seleccionar la información en las formas léxicas de la lengua origen y la lengua meta. Por ejemplo, la construcción `$2.orig.gen` se referiría al atributo `gen` de la segunda FLLO en el patrón mientras que `$2.meta.gen` se referiría al mismo atributo de la FLLM obtenida tras una consulta al diccionario bilingüe (la consulta al diccionario bilingüe está implicita en todas las acciones). La concatenación de cadenas se realiza simplemente escribiendo las expresiones de cadena una al lado de otra. Un identificador en una expresión de cadena representa el contenido de la variable de estado correspondiente.

3.3 Un ejemplo

A continuación se expone como ejemplo un fichero que contiene un regla simple para la concordancia entre nombre y determinante.

```
#DECLARACIONES
# espacios, retornos de carro y cualquier
# cosa entre [[...]] es espacio en blanco
separa :=
"([ \n\t]|\[\[([^\]]|[^\]])*]])+";

# los nombres contienen la cadena "<n>"
# seguida opcionalmente por "<acr>"
catlex nombre:="<n>(<acr>)?";

# los determinantes pueden ser subcate-
# gorizados <det><def>, <det><ind>, ...
catlex det :=
"<det>(<def>|<ind>|<dem>|<pos>)";

# nos interesan el género y el número
atribut gen:=
 {"<m>","<f>","<mf>","<GD>"};
atribut nbr:=
 {"<sg>","<pl>","<sp>","<ND>"};
# GD significa "género meta por
# determinar"; ND significa "número
# meta por determinar";

#REGLAS
# Una regla simple para concordar número
# y género entre determinante y nombre
detecta det nombre
{

# Extrae género, número y categorías
activa gen nbr det nombre;

# Hace algo solo si ambas partes
# concuerdan (forman una unidad en la LO)
si (  ( ( ($1.orig.gen!="<mf>")
          i($2.orig.gen!="<mf>")
          i($1.orig.gen==$2.orig.gen) )
        o( ($1.orig.gen=="<mf>")
          o($2.orig.gen=="<mf>") ) )
    i ( ( ($1.orig.nbr!="<sp>")
          i($2.orig.nbr!="<sp>")
          i($1.orig.nbr==$2.orig.nbr) )
        o( ($1.orig.nbr=="<sp>")
          o($2.orig.nbr=="<sp>") ) ) )

# Hace algo solo si el género o el número
# cambian en la traducción
si (  ($1.orig.gen!=$1.meta.gen)
    o ($1.orig.nbr!=$1.meta.nbr)
    o ($2.orig.gen!=$2.meta.gen)
```

```
    o ($2.orig.nbr!=$2.meta.nbr) )

# Si el género del nombre está por
# determinar, lo toma del determinante,
# si éste tiene un  género definido. Si no,
# toma el género masculino. Si ambos están
# por determinar, adoptan el masculino.
si ($2.meta.gen=="<GD>")
 {
   si (($1.meta.gen!="<mf>") i
       ($1.meta.gen!="<GD>"))
        $2.meta.gen:=$1.meta.gen
   altrament
    {
        $2.meta.gen:="<m>";
        si ($1.meta.gen=="<GD>")
            $1.meta.gen:="<m>"
    }
 }
# si no, se establece la concordancia por
# transferencia de género del nombre al
# determinante si ninguno es ambiguo.
altrament
 {
   si ($2.meta.gen!="<mf>")
    {
      si ($1.meta.gen!="<mf>")
        $1.meta.gen:=$2.meta.gen
    }
   altrament
     si ($1.meta.gen=="<GD>")
        $1.meta.gen:="<m>"
 };

# Ahora el número. Si el número en la
# LM está por determinar para el nombre
# pero está definido para el determinante,
# toma  éste a partir de aquí; si no,
# adopta el singular.
si ($2.meta.nbr=="<ND>")
 {
    si ($1.meta.nbr!="<sp>")
      $2.meta.nbr:=$1.meta.nbr
    altrament
      $2.meta.nbr:="<sg>"
 }
# Si ambos están definidos, se transfiere
# el número del nombre al determinante
altrament
 {
   si ($2.meta.nbr!="<sp>")
     si ($1.meta.nbr!="<sp>")
         $1.meta.nbr:=$2.meta.nbr
 };

# Escribe el patrón meta
envia $1.meta.lem $1.meta.det
      $1.meta.gen $1.meta.nbr &1;
envia $2.meta.lem $2.meta.nombre
      $2.meta.gen $2.meta.nbr;
```

4 El compilador

El compilador `MorphTrans` ha sido desarrollado en Linux utilizando las herramientas clásicas de construcción de compiladores `yacc` (`bison`) y `lex` (`flex`). El compilador lee un fichero fuente `MorphTrans` (`.trf`) y escribe un fichero `lex` que es convertido a continuación en un fichero en C, compilado y enlazado al programa de consulta del diccionario bilingüe para producir un programa ejecutable. El programa lex está construido de forma que cada patrón se convierte en una expresión regular que se usa para buscarlo y, como parte de la acción asociada, el patron es segmentado en formas léxicas y espacios en blanco, procesado para extraer los atributos utilizando sus definiciones regulares, y manipulado de acuerdo con código escrito por el lingüista, convenientemente convertido a C por el compilador. La consulta al diccionario bilingüe se hace a través de una función C con prototipo

```
char * Bilingue( char * ) ;
```

que toma una FLLO y proporciona una FLLM que es su equivalente designada para la lengua meta.

5 Conclusiones

Hemos presentado un lenguaje y un compilador que pueden ser utilizados para especificar e implementar la tarea de transferencia en un sistema de TA basado en la arquitectura de transferencia morfológica avanzada, siempre que las formas léxicas producidas en la fase de análisis (análisis morfológico y desambiguación léxico categorial) sean proporcionadas como cadenas. El módulo resultante consulta los lemas en un diccionario bilingüe, detecta patrones de formas léxicas de la lengua origen y las manipula de forma que son convertidas en las correspondientes secuencias de formas léxicas de la lengua meta. El lenguaje y el compilador son lo suficientemente flexibles para ser aplicados a otros sistemas y a otras lenguas origen y meta, siempre que las formas léxicas estén dadas como texto y tengan el formato *lema–categoría–información de inflexión*. Los módulos resultantes son capaces de procesar alrededor de mil formas léxicas por segundo para un par de docenas de reglas de sintagmas nominales y preposicionales. La complejidad temporal tiene dos componentes: el tiempo de detección de patrones que aumenta muy lentamente con el

número de reglas, y el tiempo de proceso de la aplicación de las reglas, el cual depende directamente de la cobertura de las reglas definidas, es decir, la frecuencia con que el código asociado a cada patrón ha de ser ejecutado porque éste ha sido detectado.

6 Trabajo futuro

En la actualidad estamos trabajando en las siguientes direcciones:

- Técnicas para identificar, a partir de córpora bilingües, posibles patrones no incluidos en el sistema.
- La optimización del código generado para la detección de las reglas con el fin de acelerar el funcionamiento del módulo.

Agradecimientos: Este trabajo es parte del proyecto interNOSTRUM, subvencionado por la Caja de Ahorros del Mediterráneo y el Vicerrectorado de Nuevas Tecnologías de la Universitat d'Alacant. También ha sido subvencionado por la Comisión Interministerial de Ciencia y Tecnología a través del proyecto TIC2000-1599-C02-02. Finalmente, queremos agradecer a Anna Esteve, lingüista en nuestro proyecto, su colaboración.

Referencias

Arnold, D. (1993). Sur la conception du transfert. En Bouillon, P. y Clas, A., editores, *La traductique*, pages 64–76. Presses Univ. Montréal, Montréal.

Arnold, D., Balkan, L., Meijer, S., Humphreys, R., y Sadler, L. (1994). *Machine Translation: An Introductory Guide*. NCC Blackwell, Oxford. Disponible en `http://clwww.essex.ac.uk/~doug/MTbook/`.

Canals-Marote, R., Esteve-Guillén, A., Garrido-Alenda, A., Guardiola-Savall, M., Iturraspe-Bellver, A., Montserrat-Buendia, S., Ortiz-Rojas, S., Pastor-Pina, H., Pérez-Antón, P., y Forcada, M. (2001). El sistema de traducción automática castellano↔catalán interNOSTRUM. En *Actas del XVII Congreso de la Sociedad Española de Procesamiento del Lenguaje Natural*.

Forcada, M. L. (2000). Learning machine translation strategies using commercial systems: discovering word-reordering rules. En *MT2000: Machine Translation and Multilingual Applications in the New Millenium (Exeter, UK, November 18–20, 2000)*, pages 7.1–7.8.

Hutchins, W. y Somers, H. (1992). *An Introduction to Machine Translation*. Academic Press.

Mira i Gimènez, M. y Forcada, M. L. (1998). Understanding PC-based machine translation systems for evaluation, teaching and reverse engineering: the treatment of noun phrases in Power Translator. *Machine Translation Review (British Computer Society)*, 7:20–27. (disponible en `http://www.dlsi.ua.es/~mlf/mtr98.ps.Z`).