PEDECIBA Informática
Instituto de Computación - Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Tesis de Maestría en Informática

Reglas Contextuales y Modelos de Estado Finito

Guillermo Moncecchi

Diciembre de 2004

Orientadora de Tesis: Dr. Ing. Dina Wonsever
Supervisora: Dr. Ing. Dina Wonsever

Resumen

Este trabajo presenta una solución para la aplicación simultánea de un subconjunto de reglas contextuales (un formalismo de reescritura para el análisis y etiquetado de segmentos de texto), basada en técnicas de estado finito. Esta solución implica la definición de un nuevo tipo de autómatas finitos, los *transductores sobre estructuras atributo-valor* y el álgebra de expresiones regulares asociada a ellos. La aplicación de las reglas sobre un texto de entrada se modela como una cascada de reemplazos basados en segmentos del texto y el contexto en que aparecen.

Palabras clave: técnicas de estado finito, transductores, expresiones regulares, reglas contextuales, autómatas sobre estructuras atributo-valor.

Agradecimientos

Este trabajo llevó tiempo. Tiempo de formación, tiempo de preparación, tiempo de elaboración. Durante ese tiempo, fueron, para mi suerte, varios y diversos quienes estuvieron ahí, ayudando, sugiriendo, apoyando.

La inspiración para el trabajo fue un curso que dictaron hace algunos años Dina Wonsever y Gustavo Crispino sobre tratamiento automático de textos, donde descubrí los transductores de estado finito. Mientras tanto, el curso de Teoría de Lenguajes, me permitió acercarme a los autómatas de todos los tipos. Quiero agradecer a Juanjo Prada, a Diego Garat, a Marcos Campal, a Tomás Laurenzo, y a Ernesto Copello por tomarse tan en serio su trabajo y permitirme con ello aprender un mucho.

El grupo de Lenguaje Natural del Instituto de Computación me permitió tener un marco donde incluir mi trabajo, y donde darle aplicación. En este tiempo, la guía de Dina fue fundamental para evolucionar hacia el resultado final. Las sugerencias de todo el equipo para mejorar lo hecho, también. La amabilidad de Ken Beesley para aclararme tantas dudas sobre relaciones regulares, transductores o lo que fuera, me allanaron varias veces el camino. A todos mi más sincero agradecimiento.

Este trabajo está dedicado a Verónica, la primera, y a mi familia, por razones obvias.

Índice general

Capítulo 1

Introducción

Las técnicas de estado finito se han utilizado desde hace mucho tiempo en la ciencia de la computación para el modelado de dominios tan diversos como el reconocimiento de patrones, manejo de circuitos digitales, análisis de sistemas operativos (redes de Petri), verificación de protocolos de comunicación o algoritmos de encriptación y compresión. Los autómatas finitos sobre los que se basan son demostradamente menos expresivos que otros formalismos, como las máquinas de Turing. Sin embargo, el utilizarlos para modelar sistemas permite aproximaciones más eficientes, precisamente por su simplicidad.

En las últimas dos décadas, han surgido nuevos resultados matemáticos y sobre todo algorítmicos en esta área, que han permitido un aumento en el uso de este tipo de técnicas (sobre la tradicional preferencia por formalismos más expresivos, como las gramáticas libres de contexto). Este tipo de resultados han sido sobre todo referidos a las relaciones regulares y los transductores de estado finito. Hoy es una referencia obligada en el área el trabajo de Kaplan&Kay sobre reglas fonológicas [12]. Allí los autores muestran que tales reglas pueden ser modeladas por lenguajes y relaciones regulares (o racionales), compilables a transductores de estado finito. Presentan en ese mismo trabajo el formalismo de reglas regulares de reescritura para la representación de tales relaciones. Posteriormente, Karttunen [13] introduce el operador de reemplazo al álgebra de expresiones regulares: este operador permite relacionar dos lenguajes, donde el segundo resulta de la sustitución de ciertas subtiras (reconocidas por expresiones regulares) en las tiras del primero, por otros lenguajes, también regulares. Este operador presenta diferentes variantes para modelar alternativas tales como el reemplazo condicionado al contexto, o el reemplazo opcional. La introducción de estos operadores ha permitido una representación en forma de reglas de las relaciones entre los lenguajes inferior y superior de una transducción, lográndose con ello una representación más clara y manejable de los diferentes fenómenos.

Paralelamente, se ha avanzado de forma importante en el desarrollo de algoritmos de obtención de transductores deterministas respecto a su entrada [17] y en la minimización de los transductores obtenidos [18], un área muy estudiada para los autómatas de estado finito [11], pero aun en desarrollo para

el caso de los transductores.

En la Lingüística Computacional, área en la que se enmarca este trabajo, el uso tradicional de las técnicas de estado finito es el del análisis morfológico y la generación. La morfología de estado finito asume que las reglas de formación de las palabras, pueden ser modeladas por medio de autómatas de estado finito [4, 15, 12, 8], y que, por lo tanto, la relación entre lemas y palabras de un lenguaje es regular. Pero, además, herramientas como los transductores han sido también utilizados en otras áreas, como análisis léxico [22], part-of-speech tagging [20] o en la representación compacta de grandes diccionarios [16]. En todos estos casos, han demostrado ser herramientas adecuadas por su capacidad de procesamiento determinista de la entrada y por permitir, en general, una representación más compacta de los datos.

El uso de máquinas de estado finito para análisis superficial (*shallow parsing*), donde lo que se intenta es recuperar sólo una parte de la información sintáctica de un texto [1], es reciente, y las aproximaciones se han basado principalmente en realizar el análisis y marcado del texto a través de cascadas de transductores de estado finito, donde cada transductor agrega información sintáctica dependiendo del contexto, pudiendo además modificar información previamente generada por los transductores de la cascada. En el trabajo de Abney donde se introduce esta técnica [2], los transductores fueron construidos directamente, pero en trabajos posteriores [3, 10] han sido generalmente compilados a partir de expresiones regulares que representan reglas de reemplazo sensibles al contexto.

Utilizando este tipo de técnicas, este trabajo pretende dar una solución para el procesamiento de un formalismo de reescritura, las *reglas contextuales*. Este formalismo, destinado al análisis y etiquetado de textos irrestrictos, es similar a las gramáticas sensibles al contexto, y por lo tanto, más poderoso que las gramáticas regulares reconocibles por máquinas de estado finito. Sin embargo, restringiendo el conjunto de reglas a aquellas que no implican una aplicación recursiva, se estudia la posibilidad de su aplicación simultánea a un texto de entrada utilizando cascadas de transductores. Esta tarea se dividirá conceptualmente en dos grandes componentes: la generación de expresiones de reemplazo regulares a partir de las reglas consideradas, y la aplicación de estas expresiones de reemplazo (y por lo tanto de las reglas contextuales originales) a un texto de entrada. La principal diferencia con trabajos anteriores, es el alfabeto sobre el que se definirán las expresiones y los transductores: en lugar de símbolos atómicos, se utilizaran *estructuras atributo-valor*, que modelan conjuntos de pares propiedad/valor, y que se relacionan por subsumción. Será entonces necesario modificar las definiciones de autómatas y transductores de estado finito para extenderlos a este nuevo alfabeto.

En el capítulo 2 se presenta el formalismo de reglas contextuales y algunas extensiones al mismo. Se describe un intérprete para su aplicación, para especificar luego el formato de un subconjunto de las reglas para las que se

intentará una aplicación utilizando técnicas de estado finito.

En el capítulo 3 se resumen primero los distintos tipos de autómatas de estado finito existentes, con sus propiedades y características más importantes, y su relación con el álgebra de expresiones regulares, intentando proporcionar una descripción del estado del arte para el área.

En el capítulo 4 se presenta primero un nuevo tipo de autómatas finitos, los *autómatas sobre estructuras atributo-valor*, y se muestra cómo modelarlos a partir de autómatas sobre predicados. A partir de este modelo teórico, se muestra el diseño de una solución al problema del trabajo, basada en la conversión de las reglas consideradas a expresiones regulares de reemplazo sobre alfabetos de estructuras atributo-valor, y su posterior aplicación en cascada sobre el texto de entrada.

En el capítulo 5 se muestran los detalles de una implementación de la solución, y algunos resultados obtenidos en su aplicación a un conjunto de reglas y textos.

Finalmente, en el capítulo 6 se destacan los principales aspectos y debilidades encontradas en la propuesta, así como algunas reflexiones respecto a la aproximación realizada, y posibles trabajos futuros basados en la misma.

Capítulo 2

Reglas Contextuales: el problema de su aplicación

2.1. El formalismo de reglas contextuales

Las reglas contextuales [25] son un formalismo de reescritura para el análisis y etiquetado de segmentos de texto. Este formalismo fue desarrollado en el marco de la metodología de exploración contextual, la cual supone que diferentes tareas de procesamiento de texto pueden resolverse analizando cierta información relevante sobre las unidades lingüísticas del texto, y teniendo en cuenta el contexto en que estas unidades aparecen. El objetivo de esta aproximación, totalmente basada en reglas, es agregar marcas a un texto de entrada, que señalen en el texto ocurrencias de fenómenos lingüísticos previamente identificados.

Este formalismo asume una representación del texto como (en lugar de una secuencia de tokens) un grafo, donde los ítems utilizados para el reconocimiento están asociados a las aristas. Estos ítems corresponden a información sobre los *tokens* del texto, u otra información que califica secuencias de tokens, ya sean agregadas por las propias reglas, o cualquier otra fuente. Los vértices del grafo de texto indican las posiciones entre las que aparecen las palabras del texto original.

Como ocurre en la mayoría de los formalismos de reescritura similares a las gramáticas sensibles al contexto, en la aplicación de reglas es posible utilizar, como parte del cuerpo o del contexto, aristas insertadas por otras reglas, o incluso por la misma regla. De esta forma es posible manejar la recursividad en la descripción de fenómenos lingüísticos.

Las reglas contextuales introducen, además, la posibilidad de especificar zonas de exclusión (estructuras que especifican la no existencia de algunos ítems en un cierto segmento de texto, de largo acotado).

Una regla contextual tiene la siguiente forma:

$$Cat \rightarrow ContextoIzquierdo \setminus Cuerpo \,/\, ContextoDerecho; EspecConj$$

donde Cat es el nombre de una categoría (que representa a los ítems antes mencionados), $ContextoIzquierdo$, $Cuerpo$ y $ContextoDerecho$ son secuencias de categorías y/o zonas de exclusión, y $EspecConj$ es una secuencia de especificaciones de conjuntos dada por extensión. Una zona de exclusión tiene el formato $*(NombreConjunto, Tam)$, donde $NombreConjunto$ es uno de los conjuntos especificados en $EspecConj$, y Tam un número natural.

Una regla contextual será aplicable a un texto de entrada, si en éste aparece la secuencia de categorías especificada en $ContextoIzquierdo$, seguida de la secuencia $Cuerpo$, seguida de la secuencia $ContextoDerecho$. En este caso, la aplicación de la regla agregará al grafo de texto una nueva arista, etiquetada con el ítem Cat,entre los vértices inicial y final de la secuencia $Cuerpo$. Las zonas de exclusión indican que, dentro del alcance dado por su largo, se admite cualquier ítem que no pertenezca a las categorías enumeradas en su especificación. En una versión extendida del formalismo, es posible especificar la aparición opcional en la secuencia de una categoría (a través del operador *op* aplicado a la categoría en la definición de la regla), o la no aparición de cierto ítem en el texto (a través del operador *no*).

Por ejemplo, para especificar que: "ante la aparición de una categoría a, precedida por una b y a una distancia no mayor a cuatro ítems (que no pertenecen a la categoría c), y seguida de las categorías d y e, agregar un arco etiquetado con el ítem f", se utiliza la regla:

$$\mathsf{f} \rightarrow \mathsf{b}\ ^{*}(\mathsf{S}, 4)\ \setminus\ \mathsf{a}\ /\ \mathsf{de}\ ; \mathsf{S} = \{\mathsf{c}\}$$

Una extensión al enunciado original del formalismo agrega poder expresivo considerando las categorías (que eran hasta ahora atómicas) como términos estructurados. Estos términos corresponden a conjuntos de pares atributo-valor, los cuales, además, están vinculados por una relación más general-más particular (subsumción). Son, entonces, ejemplos de categorías los siguientes conjuntos: $\{(\mathsf{cat}, \mathsf{verbo}), (\mathsf{persona}, 3)\}$ y $\{(\mathsf{cat}, \mathsf{verbo})\}$, y se cumple que la segunda subsume a la primera (ya que en la segunda aparecen las propiedades de la primera, con el mismo valor). Una vez incorporados estos términos, la especificación de una categoría en la regla indica la aparición de un ítem de esa categoría, o de cualquier categoría más particular que ella.

2.2. Un intérprete de reglas

Existe una implementación en Prolog de un intérprete que permite aplicar reglas contextuales a un texto.

El trabajo de este intérprete puede dividirse en tres fases:

- Carga del texto original, representado como una secuencia de palabras y sus características gramaticales en un grafo de texto, etiquetado por categorías surgidas de las características de las palabras del texto.

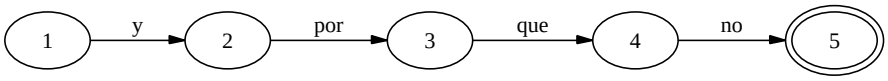

Figura 2.1: Grafo de texto

- Realización de sucesivas pasadas sobre el grafo de texto (de forma que se ve más adelante), aplicando en cada una de ellas módulos de reglas (agrupados de acuerdo al criterio de quien las escribe). En esta aplicación se agregan arcos al grafo de texto, los cuales podrán inmediatamente ser utilizados por aplicaciones subsiguientes de reglas contextuales.

- Generación de la salida a partir del grafo etiquetado

La aplicación de las reglas se realiza utilizando una estrategia de *right-corner chart parsing* [25], que recorre el grafo de texto de izquierda a derecha, analizando en cada una de las posiciones si el ítem coincide con cierta categoría distinguida (*right corner*) que toda regla en esta implementación debe especificar. En este caso, verifica la existencia en el texto del cuerpo y los contextos, y en caso de encontrarlos, agrega al grafo de texto la arista resultado de la aplicación exitosa, siguiendo luego con la recorrida del grafo. Los arcos insertados por estas reglas pueden, a su vez, lanzar la aplicación de otras reglas (lo que permite manejar la recursividad).

El intérprete también permite aplicar las reglas según un esquema de prioridades: es posible aplicar todas las reglas de un módulo al texto de entrada, o especificar que, de ser exitosa alguna regla, otras reglas no se intentarán aplicar.

Como ejemplo, supóngase que se tiene el grafo de texto de la figura 2.1, y las reglas

```
marca1 -> conj  \ por que /
marca2 -> \ que /
```

La aplicación de las reglas sin prioridad dejará el grafo como se muestra en la figura 2.2. En cambio, si se aplican en el orden en que están escritas, con prioridad, sólo se agregará el arco entre las posiciones 2 y 4

A su vez, en cada una de las modalidades, el intérprete maneja un parámetro (Vista) que indica cómo "ve" el grafo: si la vista es completa, entonces el vértice siguiente al actual es el que tiene un índice $n + 1$, si n es el índice del vértice actual. Si la vista es por categorías máximas, el vértice siguiente a un vértice dado es el extremo final del máximo arco que sale de ese nodo.

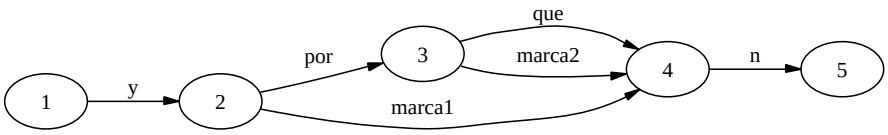

Figura 2.2: Aplicación de reglas sin prioridad

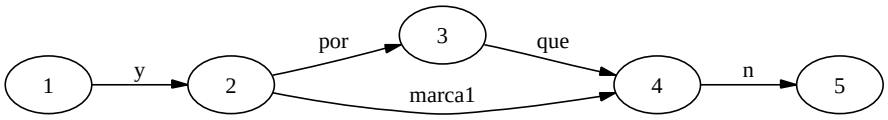

Figura 2.3: Aplicación de reglas con prioridad

2.3. Un subconjunto de las reglas contextuales y el problema

Para este trabajo, se considera un subconjunto de las reglas contextuales: aquellas en las que en su lado derecho sólo aparecen terminales, relativos a un cierto módulo. Esto es, las categorías que aparecen en lados derechos de las reglas no aparecen en el lado izquierdo de ninguna regla del módulo.

De la propia definición surge que este tipo de reglas tienen una propiedad interesante: ninguna agrega arcos al grafo de texto que puedan necesitar las otras, y no hay recursión en su aplicación. Esto lleva a pensar que es posible utilizar paradigmas de reescritura más sencillos, en particular técnicas de estado finito o modelos regulares de reglas de reemplazo, para su aplicación.

El problema a estudiar en este trabajo será el de la utilización de técnicas de estado finito para modelar la aplicación de un subconjunto de las reglas contextuales, aquellas cuyo lado derecho está compuesto sólo por terminales.

En el capítulo siguiente se presenta el marco teórico en el que se planteará la solución al problema, incluyendo un resumen del estado del arte en técnicas de estado finito.

Capítulo 3

Marco Teórico

En este capítulo se presenta el marco teórico en el que se planteará la solución al problema definido en el capítulo anterior. La sección 3.1 presenta un resumen del estado del arte en lo que a técnicas de estado finito se refiere, incluyendo los tradicionales autómatas finitos deterministas y transductores de estado finito, así como propuestas más cercanas en el tiempo como los autómatas basados en predicados. En la sección 3.2 se dan los principales conceptos sobre estructuras atributo-valor (*feature structures*), un tipo de estructura de datos que será utilizada en la solución para modelar las categorías. Como resumen del capítulo, se presentan algunas reflexiones sobre el estado del arte en el área.

3.1. Técnicas de Estado Finito

En esta sección se repasan las definiciones y principales propiedades de los autómatas de estado finito, los transductores y la definición de un álgebra de expresiones regulares asociadas a lenguajes y relaciones regulares. Se presenta, además, una extensión a los autómatas que agrega predicados asociados a las transiciones.

3.1.1. Autómatas Finitos y Lenguajes Regulares

Autómatas finitos deterministas

Citando a Hopcroft&Ullman [11],

> El autómata finito es un modelo matemático de un sistema, con entradas y salidas discretas. El sistema puede estar en cualquiera de un número finito de configuraciones internas o "estados". El estado del sistema resume la información relativa a las entradas anteriores que es necesaria para determinar el comportamiento del sistema ante subsecuentes entradas

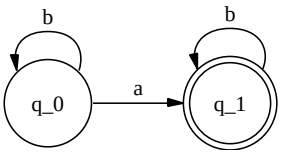

Figura 3.1: Autómata Finito Determinista

Un autómata finito puede definirse como un conjunto finito de estados y un conjunto de transiciones de un estado a otro, las cuales están asociadas a un símbolo de entrada de un alfabeto finito Σ, donde se ha definido una operación de concatenación entre tiras de símbolos.

Definición 3.1 (AFD) *Un* automata finito determinista (AFD)*[11] es una 5-tupla* $\langle Q, \Sigma, \delta, q_0, F\rangle$*, donde* Q *es un conjunto finito de* estados*,* Σ *es un* alfabeto finito de entrada*,* $q_0 \in Q$ *es el* estado inicial*,* $F \subseteq Q$ *es el conjunto de estados* finales*, y* δ *es la* función de transición *que mapea* $Q \times \Sigma$ *en* Q*.*

En lugar de utilizar la función de transición δ, se pueden de forma equivalente enumerar las transiciones del autómata por medio un conjunto E de 3-uplas, donde $\langle p, a, q\rangle \in E$, si y sólo si $\delta(p, a) = q$. A lo largo de este trabajo se utilizará, según convenga, una forma u otra para denotar las transiciones en el autómata.

Una representación gráfica común de los autómatas finitos es en forma de grafos dirigidos, donde los vértices representan a los estados, las aristas a las transiciones (etiquetadas por el símbolo asociado a la transición), y los estados finales están marcados con un círculo doble. El grafo de la figura 3.1 representa al autómata: $\langle\{q_0, q_1\}, \{a, b\}, \delta, q_0, \{q_1\}\rangle$, siendo δ dada por: $\delta(q_0, a) = q_1$, $\delta(q_0, b) = q_0$, $\delta(q_1, b) = q_1$.

Se puede extender la definición de δ a tiras del alfabeto. Sea $\hat{\delta} : Q \times \Sigma^* \to Q$. Se define

- $\hat{\delta}(q, \epsilon) = q$, y
- para toda tira w y símbolo de entrada a, $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$

En el ejemplo anterior, $\hat{\delta}(q_0, \mathsf{abb}) = q_1$.

Esta extensión conduce, concordantemente, a un conjunto extendido de aristas $\hat{E}$, donde $(p, w, q) \in \hat{E}$ si y sólo si $q = \hat{\delta}(p, w)$.

Un autómata reconoce una tira w de símbolos de su alfabeto, si $\hat{\delta}(q_0, w) \in F$. El *lenguaje reconocido por un autómata* es el conjunto de todas las tiras reconocidas por el autómata. Los lenguajes reconocidos por los autómatas finitos son llamados *lenguajes regulares.*

Existen extensiones a la definición dada de autómata finito. Una de ellas permite no determinismo en las transiciones; esto es que, partiendo de un estado, haya transiciones a dos o más estados diferentes asociadas al mismo símbolo (*Autómata Finito No Determinista* o *AFND*). A su vez, se puede permitir transiciones etiquetadas con ϵ, indicando un cambio de estado sin consumir un símbolo en la entrada (*AFND-*ϵ). Por lo tanto, la definición es la siguiente:

Definición 3.2 (AFND-ϵ) *Un* automata finito no determinista con transiciones épsilon (AFND-ϵ)*[11] es una 5-tupla* $\langle Q, \Sigma, \delta, q_0, F\rangle$*, donde* Q *es un conjunto finito de* estados*,* Σ *es un* alfabeto finito de entrada*,* $q_0 \in Q$ *es el* estado inicial*,* $F \subseteq Q$ *es el conjunto de estados* finales*, y* δ *es la* función de transición *que mapea* $Q \times \Sigma \cup \{\epsilon\}$ *en* 2^Q*.*

Si bien estas dos extensiones pueden ser útiles para modelar sistemas a través de autómatas finitos, se ha demostrado [11] que los lenguajes reconocidos por los AFND y los AFND-ϵ son los mismos que los reconocidos por los AFD. Aun más, estas demostraciones son constructivas, por lo que existen algoritmos para convertir un AFND-ϵ a un AFND, y de éste a un AFD.

Un concepto que resultará útil es el de camino en un autómata:

Definición 3.3 (Camino en un AFND) *Dado un autómata finito no determinista* $A = \langle Q, \Sigma, \delta, q_0, F\rangle$*, un* camino *de* A *es una secuencia de aristas* $(\langle p_i, a_i, q_i\rangle)_{i=1..n}$ *donde* $q_i \in \delta(p_i, a_i)$ *y* $q_i = p_{i+1}$ *para* $i = 1..n-1$*. Además, si* $p_1 = q_0$ *y* $q_n \in F$*, entonces el camino es* exitoso*.*

Es directo ver que si una tira es reconocida por un AFND, existe un camino exitoso en él, etiquetado por los símbolos de la tira.

La principal ventaja de los autómatas finitos deterministas es que proporcionan una forma efectiva de reconocer si una tira pertenece a un lenguaje regular dado. Este procedimiento es de orden lineal respecto al tamaño de la tira de entrada, e independiente de la cantidad de estados del autómata.

Expresiones regulares

Alternativamente, los lenguajes regulares pueden describirse mediante *expresiones regulares*. Las expresiones regulares sobre un alfabeto Σ y los conjuntos que denotan se definen inductivamente de la siguiente forma:

1. $\emptyset$ es una expresión regular y denota al conjunto vacío.
2. ϵ es una expresión regular y denota al conjunto $\{\epsilon\}$ formado sólo por la tira vacía.
3. Para cada a en Σ, $\boldsymbol{a}$ es una expresión regular y denota al conjunto $\{a\}$.

4. Si r y s son expresiones regulares que denotan los conjuntos R y S respectivamente, entonces $(r \mid s)$ es una expresión regular que denota $R \cup S$

5. Si r y s son expresiones regulares que denotan los conjuntos R y S respectivamente, entonces (rs) es una expresión regular que denota el conjunto resultante de la concatenación de cada tira de R con cada tira de S (concatenación de lenguajes).

6. Si r es una expresión regular que denotan al conjunto R, entonces r^* es una expresión regular que denota R^*, donde R^* se define como $\bigcup_{i=0}^{\infty} L^i$ (clausura de Kleene).

Puede demostrarse que los AFD y las expresiones regulares denotan los mismos lenguajes regulares. Una vez más esta demostración es constructiva, y permite, dada una expresión regular, construir el autómata que reconoce al lenguaje denotado por la expresión, y a la inversa.

También existen algoritmos de minimización que permiten obtener, dado un lenguaje regular, el autómata que lo reconoce con la mínima cantidad posible de estados [11].

Propiedades de los lenguajes regulares

Los lenguajes regulares tienen diversas propiedades de clausura. Son cerrados bajo las operaciones de unión, intersección, complemento, concatenación y clausura de Kleene. La principal consecuencia de esto es que es sencillo construir o definir un lenguaje regular a partir de lenguajes regulares más sencillos. Por ejemplo, el lenguaje de las tiras que comienzan con un símbolo a y siguen con una cantidad arbitraria de b's, o que comienzan con una b y continúan con cantidad arbitraria de a's es equivalente a la unión de dos lenguajes: el que contiene las tiras de la primera forma y el que contiene las tiras de la segunda forma. Cada uno de estos lenguajes, a su vez, puede verse como el resultado de la concatenación y clausura de Kleene de los lenguajes que contienen sólo al símbolo a y al símbolo b, respectivamente.

Entonces, el lenguaje puede denotarse por la expresión regular $\mathsf{ab}^* \mid \mathsf{ba}^*$. Además, el autómata que lo reconoce es el de la figura 3.2

3.1.2. Transductores y Relaciones Regulares

Transductores de Estado Finito

Los autómatas finitos vistos hasta el momento son *reconocedores* de tiras de un lenguaje. Dada una tira del alfabeto Σ, permiten saber si la tira pertenece al lenguaje reconocido por el autómata. Por lo tanto, la salida asociada a un AFD es siempre un valor binario que representa aceptar o no aceptar la tira.

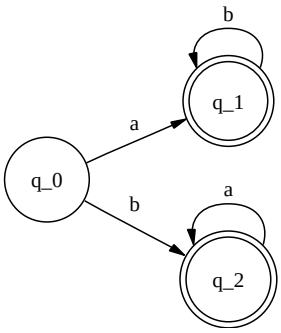

Figura 3.2: AFD generado a partir de otros

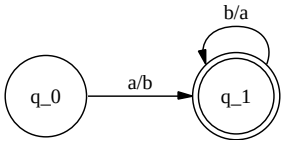

Figura 3.3: Transductor de estado finito

Una extensión a estos autómatas son los *transductores de estado finito*[1] . Estos autómatas, además de aceptar una tira, dan como salida otra tira del mismo u otro alfabeto. Para esto, cada transición tiene, además del símbolo de entrada, un símbolo de salida asociado; cada vez que el autómata está en un estado, y en la entrada aparece un símbolo dado, en caso de existir una transición que salga del estado y que tenga como etiqueta de entrada el símbolo leído, el sistema pasa al estado destino de la transición, y se escribe en la salida el símbolo de salida de la transición. Cuando una tira es reconocida por el transductor, en la salida se tendrá una tira con los símbolos de salida de las diferentes transiciones, concatenados. Por lo tanto el transductor de la figura 3.3 reconoce tiras del lenguaje denotado por la expresión regular $\mathsf{ab}*$ y devuelve tiras donde se intercambian las a's por b's y viceversa.

Definición 3.4 (FST) *Un* transductor de estado finito (FST) *[12] se define como una 5-tupla* $(Q, \Sigma_1, \Sigma_2, q_0, F, \delta)$ *, donde*

- Σ_1 *es un alfabeto finito de entrada*
- Σ_2 *es un alfabeto finito de salida*
- Q *es un conjunto finito de estados*

[1]Si bien los transductores son autómatas, cuando no exista ambigüedad, en este trabajo se utilizará el término autómata finito como sinónimo de reconocedor

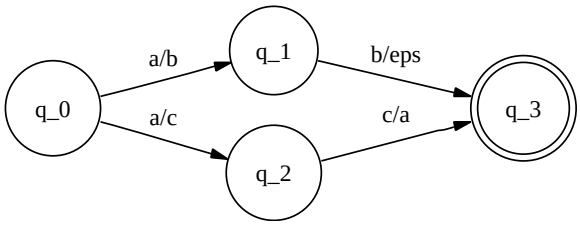

Figura 3.4: Transductor no secuencial

- q_0 *es el estado inicial*
- F *es un conjunto de estados finales*
- δ *es una función total que mapea* $Q \times \Sigma_1 \cup \{\epsilon\} \times \Sigma_2 \cup \{\epsilon\}$ *en* 2^Q

La función δ puede extenderse, igual que con los AFD, a una función $\hat{\delta} : Q \times \Sigma_1^* \times \Sigma_2^* \to 2^Q$ y una tira w de Σ_1^* será reconocida por el transductor si $\hat{\delta}(q_0, w)$ contiene un estado final.

A partir de la función δ es posible definir el concepto de camino en el transductor [21]:

Definición 3.5 (Camino en un FST) *Dado el transductor de estado finito* $T = (Q, \Sigma_1, \Sigma_2, q_0, F, \delta)$ *un* camino *de* T *es una secuencia* $(\langle p_i, a_i, b_i, q_i \rangle)_{i=1..n}$ *donde* $q_i \in \delta(p_i, a_i, b_i)$ *y* $q_i = p_{i+1}$ *para* $i = 1..n-1$*. Además, si* $p_1 = q_0$ *y* $q_n \in F$*, entonces el camino es* exitoso*.*

De la definición de $\hat{\delta}$ surge que si $q_f \in \hat{\delta}(q_0, w_1, w_2)$, entonces existe un camino exitoso $(\langle p_i, a_i, b_i, q_i \rangle)_{i=1..n}$ tal que $w_1 = a_1 \dots a_n$ y $w_2 = b_1 \dots b_n$.

Por su definición, los transductores no tienen por qué ser deterministas y admiten transiciones épsilon.

Es posible ver los transductores como autómatas finitos sobre parejas de símbolos. Si se considera a ϵ como un símbolo más, y el par (ϵ, ϵ) como el equivalente del símbolo ϵ de los AFD, todos los algoritmos válidos para los autómatas finitos siguen siendo válidos para los transductores. Sin embargo, la determinización de un transductor visto como autómata no tiene por qué generar un transductor determinista en la entrada. Por ejemplo, el transductor de la figura 3.4 es determinista visto como un autómata, pero no es determinista respecto a su entrada.[2] .

Así como los autómatas finitos definían lenguajes regulares, los transductores definen relaciones entre lenguajes regulares. La *relación regular* $R(T)$ definida por un transductor T es el conjunto de parejas $\langle x, y \rangle$ tales que $\hat{\delta}(q_0, x, y)$

[2]En las figuras el símbolo *eps* representa a ϵ

contiene un estado final. La *imagen* de una tira x bajo la relación R es el conjunto de tiras y tales que xRy. Análogamente, el *dominio* de y es el conjunto de tiras x tales que xRy.

Partiendo de una relación regular, y dada una tira del alfabeto de entrada, es posible obtener, a través de del transductor asociado, las tiras relacionadas. La relación así definida se llama *transducción*, y es por eso que los transductores pueden verse, además de como reconocedores de relaciones regulares, como computadores de transducciones.

Relaciones regulares

Las expresiones regulares pueden extenderse para denotar relaciones regulares [12]. La definición es casi idéntica a la dada para lenguajes regulares, pero en este caso se consideran, en lugar de símbolos del alfabeto, parejas de símbolos de dos alfabetos Σ_1 y Σ_2.

Entonces, las expresiones regulares forman el mínimo conjunto definido inductivamente de la siguiente forma:

1. $\emptyset$ es una expresión regular y denota a la relación regular vacía.
2. a, con $a \in \Sigma_1 \cup \{\epsilon\} \times \Sigma_2 \cup \{\epsilon\}$ es una expresión regular y denota al conjunto formado por la tira a
3. Si r y s son expresiones regulares que denotan las relaciones R y S respectivamente, entonces $(r \mid s)$ es una expresión regular que denota $R \cup S$
4. Si r y s son expresiones regulares que denotan las conjuntos R y S respectivamente, entonces (rs) es una expresión regular que denota la relación definida por $xy \mid x \in R, y \in S$, donde la concatenación de elementos de una relación es la concatenación de los correspondientes componentes.
5. Si r es una expresión regular que denotan al conjunto R, entonces r^* denota la clausura de Kleene de R.

Una extensión de las demostraciones para lenguajes regulares permite ver que la definición de las relaciones regulares como las relaciones reconocidas por un transductor es equivalente a su definición inductiva. Por lo tanto, se asume nuevamente que dado una expresión regular que denota una relación, existe un transductor que la reconoce y viceversa. O, citando a Kaplan&Kay:

> La fortaleza de nuestro método de análisis surge de la equivalencia entre estas caracterizaciones distintas. Mientras razonamos sobre relaciones regulares en términos algebraicos o de teoría de conjuntos, describimos los conjuntos en discusión por medio de expresiones regulares, y probamos las propiedades esenciales por medio de operaciones constructivas sobre los transductores de estado finito

correspondientes. Al final, por supuesto, es el transductor el que satisface nuestras necesidades computacionales.

Propiedades de clausura de las relaciones regulares

A continuación, se resumen algunas propiedades de las relaciones regulares:

- Las relaciones regulares son cerradas bajo las operación de unión. La extensión del algoritmo de unión de autómatas finito a transductores es directa.
- La imagen y el dominio de una relación regular son lenguajes regulares (considérese el autómata construido a partir de un transductor considerando solamente los símbolos de entrada o salida de las transiciones).
- Dado un lenguaje regular L, es inmediato construir un transductor que representa la relación $Id(L)$ que mapea cada tira de L en sí misma. Basta con tomar el autómata que reconoce a L y sustituir cada símbolo a de la transición por el par $\langle a, a\rangle$.
- El producto cartesiano de dos lenguages regulares es regular [12]

Composición de relaciones regulares

Se define la composición de dos relaciones R_1 y R_2 como la relación R_3 en la que, si dos tiras x y y están relacionadas por R_1, y a su vez y está relacionada con w por R_2, entonces x y w están relacionadas por la relación R_3.

Las relaciones regulares son cerradas bajo la composición. Esto se demuestra por construcción y lleva al importante resultado de que, dados dos FST, es posible obtener un transductor que representa la aplicación "en cascada" de ambos transductores.

La propiedad anterior resulta muy útil al modelar sistemas utilizando transductores, ya que permite representar diferentes transducciones y construir, finalmente, un solo transductor que aplique las transducciones en secuencia (en orden lineal respecto a la entrada, si el transductor es determinista).

Limitaciones de las relaciones regulares

A diferencia de los lenguajes regulares, las relaciones regulares *no* son cerradas bajo la operación de intersección (la relación generada por los pares $\langle x, y\rangle$ tales que xR_1y y xR_2y). Como contraejemplo, considérese las relaciones $R_1 = \{\langle a^n, b^n c^k\rangle \mid n \geq 0, k \geq 0\}$ y $R_2 = \{\langle a^n, b^k c^n\rangle \mid n \geq 0, k \geq 0\}$. Estas relaciones son regulares, ya que las expresiones regulares $(a : b)^*(\epsilon : c)^*$ y $(\epsilon : b)^*(a : c)^*$ los denotan (respectivamente). La relación intersección de ambas será $\{\langle a^n, b^n c^n\}$. Pero el rango de esta relación es el lenguaje $\{b^n c^n \mid n \geq 0\}$,

que no es regular [11]. Por lo tanto, la intersección de R_1 y R_2 no es una relación regular.

Como la intersección de conjuntos puede escribirse (utilizando las leyes de de Morgan) a partir de la unión y el complemento, y las relaciones regulares son cerradas bajo la operación de unión, entonces el hecho de que no sean cerradas bajo intersección implica que tampoco son cerradas bajo la operación de complemento.

Las relaciones de igual largo (aquellas en las que dos tiras relacionadas tienen la misma longitud), sin embargo, sí son cerradas bajo estas operaciones, y tienen la propiedad de ser reconocidas por transductores sin transiciones ϵ.

Operaciones sobre relaciones regulares

En [12], Kaplan y Kay estudiaron las principales características de los formalismos regulares de reescritura. La motivación del trabajo de Kaplan&Kay fueron los formalismos de reglas fonológicas. Este tipo de reglas tomaban un texto de entrada y devolvían el mismo texto habiendo realizado reeemplazos que dependían de partes del texto y el contexto en el que aparecían. El formato general de estas reglas era

$$\phi \rightarrow \psi \ / \lambda \ \_ \ \rho$$

Donde ϕ es la tira a ser reemplazada (o cuerpo), ψ es la tira que la reemplaza, y λ y ρ son, respectivamente los contextos izquierdo izquierdo y derecho en los que aparece el cuerpo.

El principal resultado obtenido en ese trabajo es que, si las cuatro partes de las reglas de reescritura pueden representarse por expresiones regulares,y no se permite que una regla considere como parte no contextual su propia salida, entonces la relación entre la tira original y el reemplazo puede codificarse por medio de un transductor, y por lo tanto es regular.

El *operador de reemplazo* definido en el párrafo anterior admite diferentes variantes, dependiendo principalmente de tres factores: la obligatoriedad u opcionalidad del reemplazo, el reemplazo de la tira más larga o más corta, y si el reemplazo se realiza de izquierda a derecha o de derecha a izquierda. Todas las variantes de estos operadores han sido estudiadas y documentada su construcción a partir de expresiones más sencillas, por Kartunnen [13, 14]. A continuación se enumeran las principales, con la notación definida en esos estudios, que será la utilizada en adelante en este trabajo [4]. [3]

- Reemplazo incondicional: el reemplazo incondicional de ϕ por ψ (donde ϕ y ψ son lenguajes regulares) se escribe

[3]Para una descripción completa del álgebra de expresiones regulares de Xerox, que se utilizará como notación, véase el apéndice B

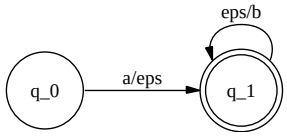

Figura 3.5: Transductor de reemplazo incondicional

$$\phi \rightarrow \psi$$

e indica que cada ocurrencia en la tira de entrada de una tira de ϕ será reemplazada por una tira de ϕ. Por ejemplo, la regla

$$\mathsf{a} \rightarrow \mathsf{b}^*$$

denota un transductor que reemplaza cada a de la tira de entrada por cero o más b's. Como puede verse, para cada tira de entrada hay infinitas tiras asociadas. El transductor que representa a esta regla es el de la figura 3.5

- Reemplazo opcional: el reemplazo opcional de ϕ por ψ se escribe

$$\phi \ (\rightarrow) \ \psi$$

e indica que, dada una tira de entrada, se obtendrá como resultado de la transducción las tiras donde cada ocurrencia de ϕ se reemplaza por una tira de ψ, o se deja como está. La regla

$$\mathsf{a} \ (\rightarrow) \ \mathsf{b}$$

mapea la tira baab en $\{\mathsf{baab}, \mathsf{bbab}, \mathsf{babb}, \mathsf{bbbb}\}$.

- Reemplazo condicional: el reemplazo condicional de ϕ por ψ con contexto izquierdo λ y contexto derecho ρ se escribe

$$\phi \rightarrow \psi \ || \ \lambda \ \_ \ \rho$$

e indica el reemplazo de ϕ por ψ en el contexto de los lenguajes λ y ρ, de forma similar a las reglas definidas por Kaplan&Kay.

Es interesante notar que en este reemplazo condicional ambos contextos se consideran *antes* del reemplazo. Existen variantes de este operador donde alguno de los contextos, o ambos, se consideran en el lenguaje resultante del reemplazo.

- Reemplazo dirigido:

  Los reemplazos vistos hasta el momento no establecen necesariamente relaciones uno a uno entre tiras, dado que ϕ y ψ son lenguajes regulares, y por lo tanto pueden tener cualquier número de tiras. Aun si ψ tiene una sola tira, se puede obtener más de una tira en la transducción dada por el reemplazo. Considérese la expresión $[\mathsf{a} \mid \mathsf{ab} \rightarrow \mathsf{c}]$: esta expresión mapea la tira ab al conjunto $\{\mathsf{cb}, \mathsf{c}\}$.

  Las ambigüedades en el reemplazo puede deberse a que una subtira de la entrada puede tener, a su vez, subtiras que son factibles de ser reemplazadas, según la misma regla. Si se reemplaza la subtira más larga, entonces las subtiras contenidas no serán reemplazadas (y a esto se llama *reemplazo longest-match*); si se reemplazan primero las subtiras, no se reemplazará la tira más larga.

  Otra fuente de ambigüedad es el orden del reemplazo. Al permitirse que un reemplazo sirva de contexto para la aplicación de otro sobre otra parte de la entrada, es posible que se obtengan resultados distintos al realizar la transducción, dependiendo del orden en que se procese la tira. Por ejemplo, la expresión $[\mathsf{a} \rightarrow \mathsf{b} \mid \_\ \mathsf{b}]$ aplicada a la tira aaab devuelve la tira aabb si el reemplazo se realiza de izquierda a derecha y la tira bbbb si el reemplazo se realiza de derecha a izquierda.

  Los operadores de reemplazo dirigido son cuatro, y no son más que la combinación de indicaciones de reemplazo *longest/shortest match* y *left-to-right* (de izquierda a derecha) o *right-to-left* (de derecha a izquierda).

  Resumiendo [4]:

  - $A@ \rightarrow B$: las tiras a reemplazar son elegidas de izquierda a derecha. Si más de una tira candidata comienza en una ubicación dada, sólo se reemplaza la más larga.
  - $A \rightarrow @B$: las tiras a reemplazar son elegidas de derecha a izquierda. Si más de una tira candidata comienza en una ubicación dada, sólo se reemplaza la más larga.
  - $A@ > B$: las tiras a reemplazar son elegidas de izquierda a derecha. Si más de una tira candidata comienza en una ubicación dada, sólo se reemplaza la más corta.
  - $A > @B$: las tiras a reemplazar son elegidas de derecha a izquierda. Si más de una tira candidata comienza en una ubicación dada, sólo se reemplaza la más corta.

  De la propia definición de estos reemplazos dirigidos, surge que la relación generada por estos operadores, en el caso que el lenguaje ψ tenga una sola tira, relaciona una tira de entrada consigo misma o con una

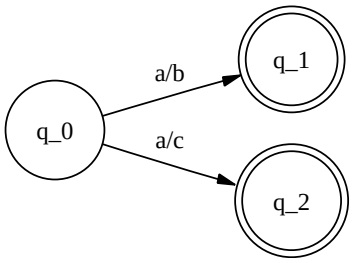

Figura 3.6: Transductor no secuencial

sola tira resultante del reemplazo. Por lo tanto, estas transducciones son funcionales, y los transductores construidos actuan como computadores de *funciones racionales* sobre tiras de Σ_1.

- Marcado

  En lugar de reemplazar una tira de entrada, se puede querer sustituir la tira por ella misma, concatenando en la salida tiras de marcado antes y despues del material a reemplazar. Este operador recibe el nombre de *markup* y la relación entre tiras dada por la aplicación del operador también es regular. Las reglas de marcado tienen la forma

  $$\phi \ \rightarrow \ \psi_1 \ \_ \ \psi_2 \ || \ \lambda \ \_ \ \rho$$

  donde ψ_1 y ψ_2 son las marcas y ϕ, λ y ρ tienen el mismo significado que para el operador de reemplazo. También se pueden extender a este operador las variantes definidas para el operador de reemplazo.

Determinización y autómatas subsecuenciales

Como se mencionó en secciones anteriores, los transductores pueden determinizarse y minimizarse con los algoritmos tradicionales, si son vistos como autómatas finitos sobre parejas de símbolos. Sin embargo, esto no quiere decir que sean deterministas respecto a su entrada. Por ejemplo, el sencillo transductor de la figura 3.6 es determinista visto como un autómata finito, pero existen dos transiciones que salen del estado 0 etiquetadas en su entrada con el símbolo *a*.

Dado que el uso de las máquinas de estado finito en general ha tenido tradicionalmente como ventaja su eficiencia al procesar la entrada en tiempo lineal respecto al tamaño de la misma, y esto es algo que surge directament del determinismo, se ha estudiado [17, 18, 16, 21] el subconjunto de los transductores que tienen, en cada estado, a lo sumo un arco de salida etiquetado con un elemento del alfabeto de entrada, a los que se ha dado en llamar transductores

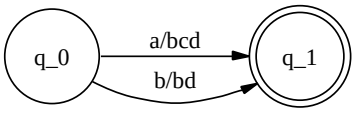

Figura 3.7: Transductor de símbolos a tiras

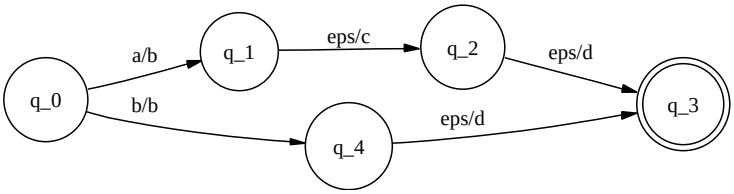

Figura 3.8: Transductor sobre símbolos equivalente

secuenciales. En esta sección se mencionan los principales resultados de esos trabajos.

Definición 3.6 (Transductor subsecuencial) *Un transductor de tipo subsecuencial T es una 7-tupla $(Q, \Sigma_1, \Sigma_2, q_0, F, \delta, \rho)$, donde*

- Σ_1,Σ_2,Q,q_0,F *se definen como en los transductores tradicionales*
- δ *es una función total que mapea* $Q \times \Sigma_1 \times \Sigma_2^*$ *en* Q
- ρ *es una función de emisión final, que mapea* F *en* Σ_2^*.

A partir de la definición, puede verse que los transductores subsecuenciales tienen algunas diferencias en su definición con los transductores en general:

- La función de de transición δ devuelve tiras de símbolos en lugar de símbolos. Esto no agrega, sin embargo, expresividad a los transductores ya que siempre es posible [21] obtener un transductor clásico a partir de un transductor con aristas etiquetadas con tiras de símbolos, dividiendo cada transición en varias, agregando estados intermedios y completando con transiciones épsilon en caso de ser necesario. El transductor de la figura 3.8 es equivalente, en este sentido, al de la figura 3.7.
- Existe una función de emisión final que permite agregar una tira de símbolos al final de la transducción. Si se agrega un símbolo # de fin de entrada al alfabeto de entrada, esta función puede reemplazarse por una transición que tiene como entrada #, y como salida la tira emitida por la función de emisión

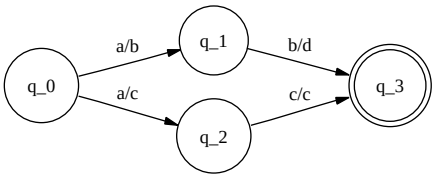

Figura 3.9: Transductor no subsecuencial

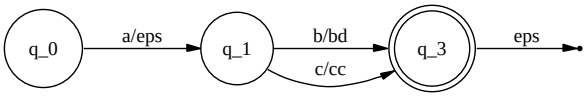

Figura 3.10: Transductor subsecuencial equivalente

- Dado un estado p y un símbolo a de entrada, existe un único estado q tal que hay una transición $(p, \langle a, b \rangle, q)$.

Por lo tanto, los transductores secuenciales son un subconjunto de los transductores de estado finito. Las funciones que pueden representarse por estos transductores son llamadas *funciones subsecuenciales*.

Existen algoritmos [17, 21] que, dado un transductor que computa una función subsecuencial, devuelven un transductor subsecuencial que computa la misma función. Previo a la aplicación de estos algoritmos es necesario, por lo tanto, determinar si el transductor original computa una función, y si esta es subsecuencial. Afortunadamente, este problema es decidible, y existen algoritmos que lo computan. Para una descripción más detallada de estos algoritmos, véase [21].

Como ejemplo, el transductor de la figura 3.10 computa la misma función que el transductor de la figura 3.9, y es, además, subsecuencial.

3.1.3. Autómatas aumentados con predicados

Muchas veces, sobre todo cuando el alfabeto es muy grande, puede ser conveniente pensar las transiciones de un autómata como correspondientes a conjuntos de símbolos más que a un símbolo del alfabeto en particular. Este conjunto podría estar definido por *extensión*, y entonces las transiciones podrían estar etiquetadas por conjuntos de símbolos que indican que la transición se toma si aparece en la entrada algún símbolo perteneciente al conjunto. También podría construirse este conjunto por *comprensión*, definiéndolo como el formado por todos aquellos símbolos del alfabeto para los cuales es verdadero un cierto *predicado lógico*. Para que esta definición sea computable, el

predicado debe tomar valores de verdad para todos los elementos del dominio, en este caso los símbolos del alfabeto del autómata. Como ejemplo, el predicado $\mathsf{not\_in}([\mathsf{a}, \mathsf{b}, \mathsf{c}])$ podría definirse como falso para los símbolos a, b y c, y verdadero para el resto de los símbolos del alfabeto.

En [24],van Noord define formalmente este tipo de autómatas y los algoritmos relacionados. A continuación se presenta un resumen de ese trabajo, que servirá como fundamento para la definición de los autómatas sobre estructuras atributo-valor.

Reconocedores con predicados

Definición 3.7 (pfsr) *Un* reconocedor de estado finito aumentado con predicados *o* pfsr M *es una 6-tupla* $(Q, \Sigma, \Pi, E, q_0, F)$*, donde* Q *es un conjunto finito de estados,* Σ *es un conjunto de símbolos,* Π *un conjunto de predicados sobre* Σ*,* E *es un conjunto finito de transiciones* $Q \times (\Pi \cup \{\epsilon\}) \times Q$*. Además,* q_0 *es un estado inicial y* $F \subseteq Q$ *es un conjunto de estados finales. Cada uno de los predicados en* Π *debe necesariamente tomar un valor de verdad para cada uno de los símbolos del alfabeto* Σ*.*

La relación $\hat{E} \subseteq Q \times \Sigma^* \times Q$ (conjunto extendido de aristas) se define inductivamente como:

1. para todo $q \in Q, (q, \epsilon, q) \in \hat{E}$,
2. para todo $(p, \epsilon, q) \in E$, $(p, \epsilon, q) \in \hat{E}$,
3. para todo $(q_0, \pi, q) \in E$ y para todo $\sigma \in \Sigma$, si $\pi(\sigma)$ entonces $(q_0, \sigma, q) \in \hat{E}$
4. si (q_0, x_1, q_1) y (q_1, x_2, q) están ambos en $\hat{E}$ entonces $(q_0, x_1 x_2, q) \in \hat{E}$

El lenguaje $L(M)$ aceptado por M se define como el conjunto

$$\{w \in \Sigma^* \mid q_f \in F, (q_0, w, q_f) \in \hat{E}$$

Al ser los predicados necesariamente verdaderos o falsos para cada uno de los símbolos, la pertenencia de una tira al lenguaje es un problema decidible. Para que este cálculo sea además eficiente, debiera garantizarse que la computación del valor de verdad de los predicados se realice en un tiempo constante, independientemente de cuál sea el símbolo sobre el cual se está evaluando.

La definición anterior puede modificarse fácilmente para que se base en una función δ de transición, como en el caso de los autómatas finitos. En este trabajo se utilizará la definición de conjunto de aristas para mantener la definición original, pero se hablará indistintamente de "conjunto de aristas" y "función de transición", y sus correspondientes versiones extendidas.

En tanto el alfabeto de entrada sea finito, los autómatas con predicados son equivalentes a los autómatas finitos "tradicionales" (basta con sustituir

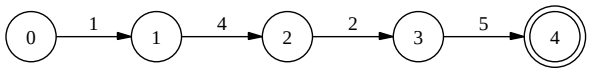

Figura 3.11: Autómata reconocedor de números

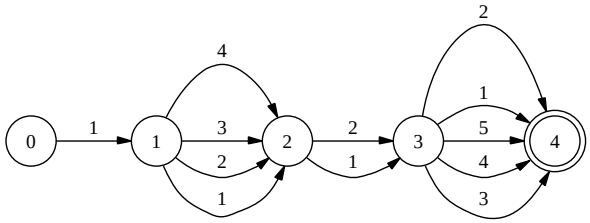

Figura 3.12: Autómata reconocedor de números menores

cada arista por un conjunto de aristas con los mismos estados origen y destino, etiquetadas con cada uno de los símbolos para los cuales el predicado es verdadero), pero en general tienen un número mucho menor de estados y transiciones.

Mientras en los autómatas tradicionales el reconocimiento de una tira se realiza exclusivamente utilizando la identidad entre sus símbolos y los símbolos de las transiciones, utilizado autómatas con predicados es posible representar otro tipo de relaciones entre los símbolos, y utilizarlas para efectuar el reconocimiento.

Como ejemplo, supóngase que se tiene un autómata basado en el alfabeto de los dígitos (como el de la figura 3.11, que reconoce el número 1425) y se pretende modificar su comportamiento para que, en lugar de reconocer las tiras cuyos símbolos pertenecen a un camino exitoso, reconozca además las tiras que representan números menores. Una posibilidad es agregar transiciones al autómata para cada uno de los símbolos que (vistos como dígitos) son menores que el que etiqueta la transición (figura 3.12). Sin embargo,se puede lograr una representación más compacta utilizando un autómata basado en el conjunto de predicados $\{\pi_i, 0 \leq i \leq 9\}$ tal que $\pi_k(n)$ es verdadero si $n \leq k$. Este autómata tendrá los mismos estados que el autómata original, y en lugar de una transición (p, k, q) existirá una transición sobre predicados (p, π_k, q) (figura 3.13).

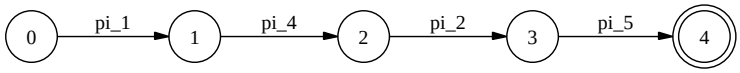

Figura 3.13: Autómata con predicados reconocedor de números menores

Operaciones sobre pfsr

Los algoritmos de unión, concatenación y clausura de Kleene usuales pueden extenderse para aplicarse a estos autómatas. Asimismo, es posible extender el algoritmo de eliminación de transiciones ϵ.

El algoritmo de intersección de los lenguajes definidos por los pfsr M_1 y M_2 construye un autómata que tiene como conjunto de estados el producto cartesiano de los estado de M_1 y M_2, y que por cada par de transiciones (p_1, π_1, q_1) en M_1 y (p_2, π_2, q_2) en M_2, define una transición $((p_1, p_2), \pi_1 \wedge \pi_2, (q_1, q_2)$ en el autómata intersección. Obsérvese que este algoritmo exige que esté definida la operación de conjunción de predicados.

La extensión del algoritmo de determinización de autómatas a pfsr plantea algunas dificultades. El algoritmo clásico mantiene conjuntos de estados y, para cada uno de estos conjuntos y símbolos del alfabeto, obtiene todos los estados a los que se llega, con transiciones etiquetadas con el símbolo, desde los estados del conjunto inicial. Cada uno de estos grupos será un estado del autómata determinista obtenido [11]. En los pfsr, las transiciones están etiquetadas con predicados y, por lo tanto, dados dos estados del pfsr y un símbolo del alfabeto, se tendrán tres grupos de estados al cual llegar: a los que se llega si ambos predicados son verdaderos, a los que se llega si el primero es verdadero y el segundo es falso, y a los que se llega si el segundo es verdadero y el primero falso. Este razonamiento puede extenderse a cualquier conjunto de estados, obteniéndose un conjunto de estados por cada combinación booleana posible de los predicados. El algoritmo definido por van Noord precisamente extiende el algoritmo de determinización calculando esos predicados, para obtener las transiciones del autómata determinista.

Como ejemplo de determinización de pfsr, considérese el pfsr trivial de la figura 3.14. La versión determinista será el pfsr que aparece en la figura 3.15.

Dos observaciones pueden hacerse respecto al algoritmo de determinización: primero, es necesario para su implementación tener definidas las operaciones booleanas de conjunción y negación de los predicados que etiquetan las transiciones. Por otra parte, las conjunciones a calcular crecen combinatoriamente con la cardinalidad del conjunto de estados, por lo que el cálculo puede ser muy costoso. Se volverá sobre esto cuando se hable de la implementación de los autómatas con predicados sobre estructuras atributo-valor.

El algoritmo de minimización de pfsr extiende de forma similar a como lo hace el de determinización, al algoritmo clásico[11]. Por una descripción

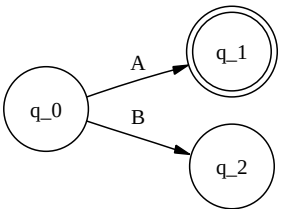

Figura 3.14: Reconocedor con predicados

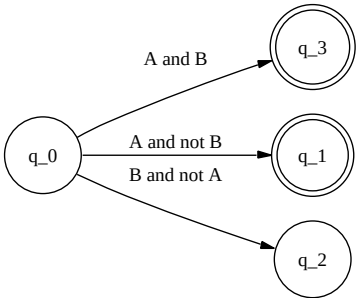

Figura 3.15: Reconocedor con predicados, determinista

completa, véase [24].

Transductores con predicados e Identidades

van Noord también extiende con predicados a los transductores definiendo los *transductores de estado finito aumentados con predicados (*pfst*)*. Estos transductores tienen como etiquetas pares de predicados indicando que dos tiras están relacionadas si sus símbolos hacen que los sucesivos predicados de un camino del transductor se satisfagan para esos símbolos (considerando los predicados de entrada para una tira y los de salida para la otra), y ese camino lleve a un estado final.

Obsérvese que, con este tipo de transductores, la imagen de la transducción computada será el conjunto de las tiras para la que la secuencia de predicados de salida son verdaderos para sus símbolos.

Definición 3.8 (pfst) *Un pfst M es una tupla $(Q, \Sigma, \Pi, E, q_0, F)$ con Q un conjunto finito de estados, Σ un conjunto finito de símbolos, Π un conjunto de predicados sobre Σ. q_0 es el estado inicial y F es un conjunto de estados finales. E es un conjunto finito $Q \times (\Pi \cup \{\epsilon\}) \times (\Pi \cup \{\epsilon\}) \times Q \times \{0, 1\}$. El componente final de las transiciones es utilizado para indicar identidades. Para todas las transiciones $(p, d, r, q, 1)$ debe ser $d = r \neq \epsilon$.*

Se define la función *str* de $\Pi \cup \{\epsilon\}$ en 2^{Σ^*}.

$$str(x) = \begin{cases} \{\epsilon\} & \text{si } x = \epsilon \\ \{s \mid s \in \Sigma, x(s)\} & \text{si } x \in \Sigma \end{cases}$$

Si $\pi \in \Pi$ es un singleton, entonces las transiciones (p, π, π, q, i) donde $i \in \{0, 1\}$ son equivalentes.

La relación $\hat{E} \subseteq Q \times \Sigma^* \times \Sigma^* \times Q$ se define inductivamente:

1. para todo p, $(p, \epsilon, \epsilon, p) \in \hat{E}$.
2. para todo $(p, d, r, q, 0), x \in str(d), y \in str(r), (p, x, y, q) \in \hat{E}$.
3. para todo $(p, \pi, \pi, q, 1)$ and $x \in str(\pi)$, $(p, x, x, q) \in \hat{E}$.
4. si las transiciones (q_o, x_1, y_1, q_1) y (q_1, x_2, y_2, q) pertenecen ambas a $\hat{E}$ entonces $(q_0, x_1x_2, y_1y_2, q) \in \hat{E}$.

La relación $R(M)$ aceptada por un pfst M se define como $\{(w_d, w_r) \mid q_f \in F, (q_0, w_d, w_r, q_f) \in \hat{E}\}$.

Estos transductores, al igual que los reconocedores, pueden transformarse a transductores clásicos y viceversa, por lo que todas las propiedades de clausura se siguen cumpliendo. Desde el punto de vista de la implementación, sin embargo, es práctico modificar los algoritmos para aplicarlos directamente a este tipo de autómatas, sin hacer la transformación.

Los pfst así definidos introducen la idea de transición *identidad*. Este tipo de transiciones indican que el símbolo de entrada y el símbolo de salida son el mismo, y sirve para definir la relación identidad(Id) que mapea tiras de un lenguaje en sí mismas y que tiene la importante característica de permitir ver a todo reconocedor como un transductor que computa la relación $Id(L)$, siendo L el lenguaje reconocido por el reconocedor.

Composición

En los transductores tradicionales, para representar la composición de relaciones, se construye un transductor basado en el producto cartesiano de los estados y se agrega una transición $((p_1, p_2), a, c, (q_1, q_2))$ si y sólo si existen transiciones (p_1, a, b, q_1) y (p_2, b, c, q_2) en los transductores que representan las relaciones a componer (en ese orden).

El algoritmo de composición de pfst es similar, pero, en lugar de exigir que el símbolo de entrada de M_1 sea igual al de entrada de M_2 (siendo M_1 y M_2 los transductores a componer), exige que se satisfaga la conjunción del predicado de salida de M_1 con el de entrada de M_2. Además, debe extenderse para contemplar los casos que uno de los lados de la transición es ϵ y para manejar identidades. Para una descripción completa del algoritmo, véase el trabajo de van Noord.

Determinización

Un transductor con predicados es determinista [24] si no hay estados $p, q \in Q$ tales que $(p, \epsilon, x, q, i) \in E$ y si para todos los estados p y símbolos σ existe a lo sumo una transición (p, π_d, x, q_i) tal que $pi_d(\sigma)$.

Para extender los algoritmos de determinización de transductores, se extiende la definición de transductor de forma que la parte de salida de una transición es una secuencia de predicados. Sobre esta definición es posible extender los algoritmos de eliminación de transiciones ϵ [21] y secuencialización [16].

Los transductores sobre predicados introducen, además, un problema adicional en la determinización, por la existencia de identidades entre símbolos de entrada y salida. Existe una versión extendida de los pfst, los pfst con cola acotada (*pfst with a bounded Queue*), que se describen en [24], y que permiten resolver el problema.

En resumen, los algoritmos de determinización aplicables a transductores de estado finito se han extendido (introduciendo algunas modificaciones al paradigma) a los transductores sobre predicados. Por supuesto, esta extensión mantiene el problema de la existencia de transductores no secuencializables.

3.1.4. Estado del arte

En las secciones previas se ha repasado el estado del arte en lo que a técnicas y formalismos de estado finito se refiere. Para los autómatas de estado finito reconocedores, existen estudios muy exhaustivos y completos al respecto, desde la especificación de los autómatas y algoritmos para manipularlos, pasando por la relación con los lenguajes regulares y las diferentes propiedades de clausura, hasta el alcance de su expresividad. En el área de las relaciones regulares y los transductores no parece darse la misma realidad, si bien los avances realizados en los últimos años han sido notables. Si bien el formalismo de las relaciones racionales se remonta a mucho tiempo atrás, desde el punto de vista de su estudio computacional, los trabajos son casi todos de la década de los noventa, y es posible que por eso no exista aun una especificación estándar de los transductores (por ejemplo, compárense las definiciones de transductores y de transductores subsecuenciales). Parecen existir dos grandes líneas de investigación en paralelo: una (donde el principal exponente sea probablemente Lauri Karttunnen) que se basa en el estudio de las expresiones regulares para el modelado de reemplazos y su relación con los transductores, y otra (Mehryar Mohri) que presenta una aproximación más directa a los transductores en sí, especificando algoritmos de determinización y minimización.

3.2. Estructuras Atributo-Valor

En esta sección se presenta un tipo de estructuras de datos llamadas *estructuras atributo-valor* (feature structures) que modelan conjuntos de parejas de propiedades y sus respectivos valores. Estas estructuras servirán para modelar las etiquetas en el problema de este trabajo, y serán los símbolos del alfabeto para los fs-autómatas, presentados más adelante.

3.2.1. Definición

Las estructuras atributo-valor (EAV) son estructuras de datos genéricas que representan conjuntos de propiedades con valores que pueden ser atómicos u otras EAVs. Por su generalidad, han sido utilizadas por diversos formalismos de análisis de lenguaje natural como forma de representar distintos componentes del texto. Uno de los usos más populares de las EAVs ha sido en el marco de las gramáticas HPSG (Head-Driven Phrase Structure Grammar) [19].

Estas estructuras pueden representarse como matrices de propiedad-valor. Por ejemplo, una EAV que indica que la propiedad CAT tiene el valor verbo, y la propiedad AGR tiene como valor otra EAV que representa las parejas de propiedades y valores $\{\mathsf{NUMERO} : \mathsf{sg}, \mathsf{PERSONA} : 3\}$, se representa de la siguiente forma

$$\begin{bmatrix} CAT & np \\ AGR & \begin{bmatrix} NUMERO & sg \\ PERSONA & 3 \end{bmatrix} \end{bmatrix}$$

Es posible que varios atributos de una EAV compartan un valor. Esta propiedad se conoce como *reentrancia* y se representa en las matrices atributo-valor numerando la primera aparición del valor, y poniendo como valor el número asignado, en las siguientes propiedades que compartan el valor dentro de la estructura.

$$\begin{bmatrix} CAT & s \\ HEAD & \begin{bmatrix} AGR & \boxed{1}\begin{bmatrix} NUMERO & sg \\ PERSONA & 3 \end{bmatrix} \\ SUBJ & [\ AGR\ \ \boxed{1}\ ] \end{bmatrix} \end{bmatrix}$$

3.2.2. Subsumción y Unificación

Se dice que una EAV *subsume* a otra, cuando la segunda contiene (al menos) tanta información como la primera. Esto es, toda propiedad definida en la primera estará también definida en la segunda, y tendrá el mismo valor, sin perjuicio de que la segunda tenga definidas otras propiedades que agregan información sobre el objeto que describe. Es decir, la subsumción (denotada $\sqsubseteq$)

relaciona las estructuras en un orden más general- más particular. Claramente, es posible que dos estructuras dadas no estén relacionadas por subsumción; en este caso se dice que estas EAVs son *incompatibles*.

Las EAVs que siguen están relacionadas, ya que la segunda subsume a la primera.

$$\begin{bmatrix} CAT & s \\ HEAD & \begin{bmatrix} AGR & \boxed{1}\begin{bmatrix} NUMERO & sg \\ PERSONA & 3 \end{bmatrix} \\ SUBJ & [\ AGR\ \ \boxed{1}\ ] \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} CAT & s \\ HEAD & \begin{bmatrix} AGR & \begin{bmatrix} NUMERO & sg \\ PERSONA & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Dadas dos o más EAVs, existe una función parcial llamada *unificación* (denotada $\sqcup$) que obtiene la estructura más general que contiene toda la información de las originales. O, equivalentemente, la estructura más general que es subsumida por ambas.

Por ejemplo,

$$[\ CAT\ \ verbo\ ]\ \sqcup\ [\ TIEMPO\ \ pres\ ]\ =\ \begin{bmatrix} CAT & verbo \\ TIEMPO & pres \end{bmatrix}$$

La operación de unificación es una función parcial, ya que en caso de ser las estructuras incompatibles, el resultado no está definido. La estructura vacía ([ ]) representa al conjunto vacío y actúa como elemento identidad para la operación de unificación, esto es $F \sqcup [\ ] = [\ ] \sqcup F = F$ para toda estructura F.

En [6, 7], Carpenter define formalmente las estructuras atributo-valor, e introduce una lógica sobre estructuras tipadas. La definición de Carpenter es más general que la manejada en las secciones anteriores, ya que asume un tipo para cada estructura, lo que hace que la relación de subsumción no se defina sólo por relación entre las propiedades, sino incluso entre los valores que toman y los tipos que se le asocian. Para trabajo, sin embargo, sólo interesarán las estructuras atributo-valor no tipadas, con valores atómicos y sin reentrancia

Capítulo 4

Un modelo para la aplicación de reglas contextuales

Considérese el enunciado del problema dado en el capítulo 2:

> El problema a estudiar en este trabajo será el de la utilización de técnicas de estado finito para la aplicación de un subconjunto de las reglas contextuales, aquellas cuyo lado derecho está compuesto sólo por terminales.

El dominio del problema se ha restringido, entonces, a un subconjunto de las reglas contextuales [1] . Estas reglas, por lo tanto, pueden considerarse con el formato:

$$Cat \rightarrow CIzq \setminus Cuerpo \,/\, CDer; EspecConj$$

donde en el lado derecho no aparece ninguna categoría que figure como lado izquierdo de otra regla (ni, por supuesto, de la misma regla). En la especificación original [25] tanto el contexto izquierdo, como el cuerpo y el contexto derecho son secuencias opcionales de categorías atómicas o zonas de exclusión. Se verá que los lados derechos así especificados pueden modelarse a través de expresiones regulares y que la identificación en el grafo de texto puede verse como una sucesión de reemplazos utilizando el paradigma de relaciones regulares de reescritura. La utilización de tal paradigma permitirá la aplicación simultánea de un conjunto de reglas.

En este capítulo se presenta una solución al problema mencionado, basada en esta idea. En una primera aproximación se da una solución que contempla el formato original de las reglas, para luego extender la idea a los casos en los que las categorías son términos compuestos y (posiblemente) subespecificados. Se sugiere luego una modificación al intérprete y su diseño. Previo a la presentación de la solución, es necesario modificar el marco teórico, introduciendo un nuevo tipo de autómatas de estado finito, los *autómatas sobre*

[1] en adelante, cuando se hable de "reglas contextuales" se estará haciendo referencia al subconjunto especificado

estructuras atributo-valor, que serán necesarios para el diseño de la solución definitiva. mostrando además que es posible modelarlo como un subconjunto de los autómatas sobre predicados.

4.1. Autómatas sobre estructuras atributo-valor

Para poder resolver el problema de este trabajo, fue necesario extender los autómatas y transductores para que operen sobre un alfabeto más complejos que los alfabetos atómicos tradicionales. En esta sección se presenta un nuevo tipo de autómata finito, el cual se basa en un alfabeto compuesto por estructuras atributo-valor, y en el que la modificación del estado se produce si el símbolo en la entrada se encuentra comprendido (por la relación de subsumción) entre dos símbolos que etiquetan una transición de salida del estado actual. Es decir, la relación de subsumción sustituye en estos autómatas a la de identidad, utilizada en los autómatas clásicos. La definición de estos autómatas conlleva la extensión del álgebra de expresiones regulares para modelarlos.

4.1.1. Definición

Los autómatas sobre estructuras atributo-valor son autómatas finitos similares a los AFND que se describieron en el capítulo anterior, pero con algunas características particulares:

- El alfabeto es un conjunto de estructuras atributo-valor, entre las cuales existe (por definición) una relación de subsumción que indica que un símbolo es más general que otro. En este alfabeto se ha definido una operación de concatenación, que permite definir *tiras* de estructuras, y existen estructuras $\bot$ (bottom), y $\top$ (top) que son, respectivamente, más general y más particular que cualquier estructura del alfabeto.
- La función de transición se define de forma similar que para los AFND, pero asociada a pares de símbolos, y la definición del reconocimiento especifica que una tira es reconocida si existe un camino que parte del estado inicial, llega a un estado final, y donde los primeros símbolos de las transiciones subsumen al símbolo correspondiente de la tira de entrada, mientras los segundos son subsumidos por dicho símbolo.

Por ejemplo, el autómata de la figura 4.1 está definido sobre el alfabeto de EAVs en las que las propiedades son {cat,numero,genero}, con valores {det, nombre} para cat, {singular, plural} para numero, y {fem, masc} para genero. Este autómata reconoce las tiras del alfabeto cuyo primer símbolo es la estructura [cat : det, numero : plural] o una estructura más particular, y el segundo es [cat:nombre,numero:plural] o una estructura más particular. Por lo

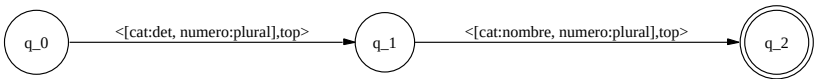

Figura 4.1: fs-autómata

tanto, la tira "[cat:det,numero:plural,genero:fem] [cat:nombre,numero:plural, genero:masc]" es reconocida, y la tira "[cat:det] [cat:nombre,numero:plural]" no lo es.

Definición 4.1 (fs-autómata) *Un* autómata sobre estructuras atributo-valor (fs-automata) *es una 5-tupla* $(Q, \Sigma, q_0, F, \delta)$ *donde* Σ *es un alfabeto finito de estructuras atributo-valor (donde se asume la existencia de un elemento* $\bot$ *que es más general que cualquier estructura del conjunto, y un elemento* $\top$ *que es más particular),* Q *es un conjunto finito de* estados*,* q_0 *es el* estado inicial*,* F *es un conjunto de* estados finales*, y* δ *es una* función de transición *de* $Q \times (\Sigma \times \Sigma)^\epsilon$ *en* 2^Q*, y que cumple que si* $\delta(q, a, b)$ *está definida, entonces* $a \sqsubseteq b$*.*

Se puede extender la función δ a tiras de EAVs definiendo $\hat{\delta} : Q \times \Sigma^* \rightarrow 2^Q$ tal que

$$\hat{\delta}(q, aw) = \bigcup \delta(q', w), \forall q' \in Q', \textit{siendo } Q' = \{q' \mid q' \in \delta(q, b, c) \wedge b \sqsubseteq a \wedge a \sqsubseteq c\}$$

El lenguaje $L(M)$ aceptado por un fs-autómata M es el conjunto $\{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$.

Este tipo de autómatas es (en su definición inicial) no determinista, y con transiciones ϵ. Para la eliminación de transiciones ϵ puede extenderse el algoritmo clásico para autómatas finitos. Sin embargo, para la determinización debe tenerse en cuenta la modificación de la función de transición extendida, ya que podrían existir pares de transiciones etiquetadas con estructuras diferentes (y por lo tanto consideradas distintas por el algoritmo clásico), que resultaran en el mismo estado para un mismo símbolo de entrada, partiendo de un cierto estado. Esto es, si, por ejemplo, existieran en un autómata dado estructuras atributo-valor a y b que cumplieran:

$$\begin{aligned} \delta(q_0, a, \top) &= \{q_1\} \\ \delta(q_0, b, \top) &= \{q_2\} \end{aligned}$$

Si $a \sqsubseteq c$ y $b \sqsubseteq c$, entonces $\hat{\delta}(q_0, c) = \{q_1, q_2\}$.

También deben modificarse en estos autómatas las operaciones sobre ellos, para que tengan en cuenta que las transiciones están etiquetadas no con un único símbolo, sino con un conjunto de ellos.

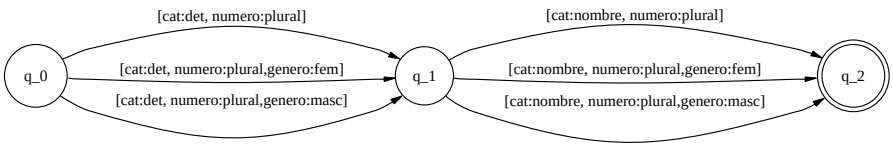

Figura 4.2: AFND equivalente

Se verá que estos problemas (el de la determinización y el de la modificación de las operaciones) ya están resueltos si se observa que los fs-autómatas pueden representarse como un caso particular de autómatas aumentados con predicados.

Dado un fs-autómata se puede construir un autómata finito sobre el mismo alfabeto, con los mismos estados que el original, y donde, por cada transición del autómata original, se agregan transiciones etiquetadas con los símbolos comprendidos por la relación de subsumción entre los símbolos que etiquetan la transición. Este autómata reconocerá el mismo lenguaje que el fs-autómata original. Por ejemplo, el autómata finito de la figura 4.2 reconoce el mismo lenguaje que el fs-autómata de la figura 4.1.

Previo a demostrar formalmente esta propiedad, se introducen los conceptos de subsumción entre tiras de estructuras atributo-valor, y de camino exitoso en un fs-autómata.

Definición 4.2 (Subsumción de tiras) *Si F es un alfabeto cuyos símbolos son estructuras atributo-valor, se define la relación de* subsumción entre tiras*, denotada por $\sqsubseteq$, como el menor subconjunto de $F^* \times F^*$ que cumple:*

- $\epsilon \sqsubseteq \epsilon$
- *si $w \sqsubseteq w'$ y $a \sqsubseteq a'$, con $a, a' \in F$, y $w, w' \in F^*$ entonces $wa \sqsubseteq wa'$.*

Para los fs-autómatas, la definición de camino es similar a la dada para los autómatas finitos no deterministas:

Definición 4.3 (Camino en un fs-autómata) *Dado un fs-autómata $A = (Q, \Sigma, q_0, F, \delta)$, un* camino *de A es una secuencia $(\langle p_i, a_i, b_i, q_i\rangle)_{i=1..n}$ donde $q_i \in \delta(p_i, a_i, b_i)$ y $q_i = p_{i+1}$ para $i = 1..n-1$. Además, si $p_1 = q_0$ y $q_n \in F$, entonces el camino es* exitoso*.*

Sin embargo, en este caso cambia la relación entre camino exitoso y la función $\hat{\delta}$, ya que si $q_f \in \hat{\delta}(q_0, w)$, entonces necesariamente existe un camino exitoso $(\langle p_i, a_i, b_i, q_i\rangle)_{i=1..n}$ tal que $a_1 \dots a_n \sqsubseteq w$ y $w \sqsubseteq b_1 \dots b_n$.

Los teoremas que siguen muestran que los fs-autómatas reconocen exactamente a los lenguajes regulares definido sobre un conjunto finito de estructuras atributo-valor.

Teorema 4.4 Dado un fs-autómata $A = (Q, \Sigma, q_0, F, \delta)$, existe un autómata finito no determinista $A' = (Q, \Sigma, q_0, F, \delta')$ que reconoce al mismo lenguaje que A.

Para demostrarlo, se define δ' de la siguiente forma:

- si $q \in \delta(p, \epsilon)$, entonces $q \in \delta'(p, \epsilon)$
- si $q \in \delta(p, a, b)$, con $a, b \in \Sigma$, entonces $q \in \delta'(p, a')$, para todo $a' \in \Sigma$ que cumple $a \sqsubseteq a'$ y $a' \sqsubseteq b$.

y se muestra que $w \in L(A)$ sii $w \in L(A')$.
Previamente, se demuestra el siguiente lema:

Lema 4.5 Existe en A un camino $\pi = (\langle p_i, a_i, b_i, q_i\rangle)_{i=1..n}$ donde $a_1 \dots a_n \sqsubseteq w$ y $w \sqsubseteq b_1 \dots b_n$, si y sólo si existe en A' un camino $\pi' = (\langle p_i, a'_i, q_i\rangle)_{i=1..n}$ con $a'_1 \dots a'_n = w$.

La demostración en un sentido se hace aplicando el principio de inducción completa sobre el largo de π.

Si el largo de π es 0, el resultado se cumple trivialmente. Supuesto que para todo camino π de largo menor que un cierto h mayor que 0 el lema se cumple, considérese un camino $\pi_h = (\langle p_i, a_i, b_i, q_i\rangle)_{i=1..h}$, con $a_1 \dots a_n \sqsubseteq w$ y $w \sqsubseteq b_1 \dots b_n$. Por definición de camino, existe en A un camino $\pi_{h-1} = (\langle p_i, a_i, b_i, q_i\rangle)_{i=1..h-1}$ con $a_1 \dots a_{h-1} \sqsubseteq w'$ y $w' \sqsubseteq b_1 \dots b_{h-1}$, siendo $w = w'a$, con $a \in \Sigma$. Por la hipótesis inductiva, existe en A' el camino $\pi'_{h-1} = (\langle p_i, a'_i, q_i\rangle)_{i=1..h-1}$, con $a'_1 \dots a'_{h-1} = w'$. Como $q_h \in \delta(q_{h-1}, b, c)$ con $b \sqsubseteq a$ y $a \sqsubseteq c$, debe cumplirse, por definición, que $q_h \in \delta'(q_{h-1}, a)$, y por lo tanto existe en A' un camino $\pi' = (\langle p_i, a'_i, q_i\rangle)_{i=1..h}$, donde $a'_1 \dots a_h = w'a = w$.

La demostración en el otro sentido es análoga, realizando inducción completa sobre el largo dr π'.

Si el largo de π' es 0, el resultado se cumple trivialmente. Supuesto que para todo camino π' de largo menor que un cierto h mayor que 0 el lema se cumple, considérese un camino $\pi'_h = (\langle p_i, a'_i, q_i\rangle)_{i=1..h}$, con $a'_1 \dots a'_n = w$. Por definición de camino, existe en A' un camino $\pi'_{h-1} = (\langle p_i, a'_i, q_i\rangle)_{i=1..h-1}$ con $a_1 \dots a_{h-1} = w'$, siendo $w = w'a$, con $a \in \Sigma$. Por la hipótesis inductiva, existe en A el camino $\pi_{h-1} = (\langle p_i, a_i, b_i, q_i\rangle)_{i=1..h-1}$, con $a_1 \dots a_{h-1} \sqsubseteq w'$ y $w' \sqsubseteq b_1 \dots b_{h-1}$. Como $q_h \in \delta'(q_{h-1}, a)$, debe cumplirse, por definición, que $q_h \in \delta(q_{h-1}, b, c)$, siendo $b \sqsubseteq a$ y $a \sqsubseteq c$ y por lo tanto existe en A un camino $\pi = (\langle p_i, a_i, b_i, q_i\rangle)_{i=1..h}$, donde $a_1 \dots a_h = a_1 \dots a_{h-1}b \sqsubseteq w$ y $w \sqsubseteq b_1 \dots b_h = b_1 \dots b_{h-1}c$.

Demostrado el lema, es directo ver que $w \in L(A)$, existe en A un camino exitoso $\pi = (\langle p_i, a_i, b_i, q_i\rangle)_{i=1..n}$ donde $a_1 \dots a_n \sqsubseteq w$ y $w \sqsubseteq b_1 \dots b_n$, y por lo tanto existe un camino exitoso $\pi' = (\langle p_i, a'_i, q_i\rangle)_{i=1..n}$ en A', donde $a'_1 \dots a'_n = w$, y por lo tanto $w \in L(A')$. Análogamente, si $w \in L(A')$, entonces $w \in L(A)$.

Teorema 4.6 Dado un AFND $A = (Q, \Sigma, q_0, F, \delta)$, existe un fs-autómata $A' = (Q, \Sigma, q_0, F, \delta')$ que reconoce al mismo lenguaje que A.

La demostración de este teorema es prácticamente trivial, si se considera que un autómata finito no determinista puede verse como un fs-autómata en el que en cada transición se reconoce el conjunto de símbolos subsumidos y que subsumen a un símbolo dado, conjunto que sólo incluye al propio símbolo.

Por lo tanto, dado A, se define δ' de la siguiente forma:

- si $q \in \delta(p, \epsilon)$, entonces $q \in \delta'(p, \epsilon)$
- si $q \in \delta(p, a)$, con $a \in \Sigma$, entonces $q \in \delta'(p, a, a)$.

Y se demuestra el siguiente lema:

Lema 4.7 Existe en A un camino $\pi = (\langle p_i, a_i, q_i \rangle)_{i=1..n}$ donde $a_1 \dots a_n = w$, si y sólo si existe en A' un camino $\pi' = (\langle p_i, a_i, a_i, q_i \rangle)_{i=1..n}$.

La demostración en un sentido se hace por inducción completa en el largo de π.

Si el largo de π es 0, el resultado se cumple trivialmente. Supuesto que para todo camino π de largo menor que un cierto h mayor que 0 el lema se cumple, considérese un camino $\pi_h = (\langle p_i, a_i, q_i \rangle)_{i=1..h}$, con $a_1 \dots a_n = w$. Por definición de camino, existe en A un camino $\pi_{h-1} = (\langle p_i, a_i, q_i \rangle)_{i=1..h-1}$ con $a_1 \dots a_{h-1} = w'$, siendo $w = w'a_h$. Por la hipótesis inductiva, existe en A' el camino $\pi_{h-1} = (\langle p_i, a_i, a_i, q_i \rangle)_{i=1..h-1}$. Como $q_h \in \delta(q_{h-1}, a_h)$, debe cumplirse, por definición, que $q_h \in \delta'(q_{h-1}, a_h, a_h)$ por lo tanto existe en A' un camino $\pi' = (\langle p_i, a_i, a_i, q_i \rangle)_{i=1..h}$.

Análogamente, la demostración en el otro sentido se hace por inducción completa en el largo de π'.

Demostrado el lema, y de forma análoga al teorema anterior, se prueba que $w \in L(A)$, entonces $w \in L(A)$, y a la inversa.

El resultado anterior implica que todo lenguaje que puede representarse con un fs-autómata puede representarse con un autómata con alfabeto compuesto por EAVs, por lo que los fs-autómatas no agregan poder expresivo a los AFND. Sin embargo, en la práctica, muchos problemas (como el que motiva este trabajo) se modelan de forma más natural considerando una relación de orden parcial entre los símbolos del alfabeto (como las que relacionan a estas estructuras). Además, la representación puede ser más compacta. Estos argumentos son muy similares a los dados en la presentación de los autómatas con predicados; en la próxima sección se verá que los fs-autómatas pueden representarse de forma natural como pfsr donde los predicados son combinaciones booleanas de estructuras atributo-valor.

4.1.2. fs-autómatas como pfsr

Definida la relación de subsumción entre las estructuras que componen el alfabeto, los fs-autómatas pueden verse como autómatas finitos con transiciones sobre conjuntos de símbolos. En cada transición, este conjunto está dado por aquellos símbolos subsumidos por el primer símbolo de la pareja que etiqueta la transición, y que subsumen al segundo. Por esto, es posible verlos como un subconjunto de los autómatas finitos con predicados que se vieron en la sección 3.1.3, donde los predicados que etiquetan las transiciones son verdaderos para un símbolo dado, si el símbolo pertenece al conjunto definido anteriormente, y falsos de otra forma.

Una transición (q_1, a, b, q_2) en un fs-autómata se codificará como (q_1, π, q_2) en el pfsr correspondiente, donde π se satisface para todos los símbolos subsumidos por a, y que subsumen a b, y sólo por ellos. Los estados serán, además, los mismos que para el fs-autómata. Para lograr esto, los predicados se definirán como combinaciones booleanas sobre predicados que son verdaderos evaluados para las estructuras subsumidas por una estructura dada, y falsos en otro caso. Por lo tanto, una transición $[p, a, b, q]$ en un fs-autómata, se representará por $[p, a \wedge \bar{c}, q]$, siendo a y c predicados que evalúan como verdaderos para los símbolos del alfabeto más particulares que a y c respectivamente, y siendo c la estructura más general del alfabeto que es más particular que b, o $\top$ en caso de ser $b = \top$.

A continuación se definen los fs-autómatas vistos como pfsr. Previamente, se definen los predicados en los que se basarán.

Definición 4.8 (fs-predicados) *Dado un conjunto Σ de estructuras atributo-valor, los fs-predicados son el mínimo conjunto de predicados definidos inductivamente de la siguiente forma:*

- *Si $a \in \Sigma$ entonces a es un predicado que se satisface para todo símbolo más particular que a.*
- *Si $a \in \Sigma$ entonces $\bar{a}$ es un predicado que se satisface para un símbolo si a es falso para ese símbolo. Se llama a los predicados a y $\bar{a}$* literales
- *Si $a_1 \ldots a_n$ son literales, entonces $a_1 \wedge \ldots \wedge a_n$ es un predicado (conjunción) que se satisface si todos los literales involucrados son verdaderos*
- *Si $a_1 \ldots a_n$ son literales o conjunciones, entonces $a_1 \vee \ldots \vee a_n$ es un predicado (disyunción) que se satisface si uno de los literales o conjunciones involucrados es verdadero*

Como ejemplo, considérese el conjunto de EAVs siguiente:

$$\{[\mathsf{cat} : \mathsf{verbo}, \mathsf{modo} : \mathsf{ind}], [\mathsf{cat} : \mathsf{verbo}], [\mathsf{modo} : \mathsf{ind}], [\mathsf{cat} : \mathsf{nombre}]\}$$

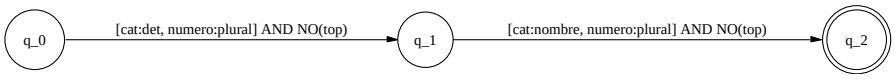

Figura 4.3: fs-pfsr equivalente a un fs-autómata

Son fs-predicados verdaderos evaluados para la estructura

$$[\mathsf{cat} : \mathsf{verbo}, \mathsf{modo} : \mathsf{ind}]$$

los predicados $[\mathsf{cat} : \mathsf{verbo}]$, $[\mathsf{cat} : \mathsf{verbo}] \wedge [\mathsf{modo} : \mathsf{ind}]$, $(\overline{[\mathsf{cat} : \mathsf{nombre}]})$.

Por la definición de los predicados, dado cualquier símbolo de Σ, un predicado será necesariamente verdadero o falso para ese símbolo.

Definición 4.9 (fs-pfsr) *Un* autómata finito extendido con predicados sobre estructuras atributo-valor (fs-pfsr) *es una 6-tupla* $(Q, \Sigma, \Pi, E, q_0, F)$*, donde* Q *es un conjunto finito de estados,* Σ *es un conjunto finito de EAVs (donde se asume la existencia de un elemento* $\bot$ *que es más general que cualquier estructura del conjunto, y un elemento* $\top$ *que es más particular),* E *es un conjunto finito de transiciones* $Q \times (\Pi \cup \{\epsilon\}) \times Q$*.* q_0 *es el estado inicial y* $F \subseteq Q$ *es un conjunto de estados finales.* Π *son los fs-predicados sobre* Σ*.*

El fs-pfsr de la figura 4.3 reconoce el mismo lenguaje que el fs-autómata de la figura 4.1.

Esta representación de los fs-autómatas tiene la ventaja que tienen todas las aproximaciones donde un problema se considera como un caso particular de otro: las propiedades y algoritmos definidos para el caso más general estarán definidos para el más particular. En este caso, todas las operaciones sobre autómatas (unión, concatenación, complemento, clausura de Kleene) están definidos para los autómatas con predicados, por lo que la extensión del álgebra de expresiones regulares es directa, sustituyendo las operaciones originales por las definidas para pfsr.

Adicionalmente, los predicados permiten modelar de forma más compacta transiciones sobre conjuntos de símbolos definidos como complemento de otros. Por ejemplo, considérese el fs-autómata que reconoce estructuras que *no* son de la forma a, siendo a una EAV, representado en la figura 4.4, y basado en el alfabeto $\{b, a, c, d, e\}$, donde $b \sqsubseteq a \sqsubseteq c$ y $d \sqsubseteq e$. Puede verse que es necesario, para su construcción, determinar el símbolo más particular que subsume a a, así como el más general que es subsumido por él (en este caso, b y c respectivamente), para generar los conjuntos de símbolos más particulares y más generales que a. Además, debe crearse una transición por cada conjunto de símbolos que no está relacionado por subsumción con a. En la versión con predicados (figura 4.5), en cambio, basta definir dos predicados: uno que es

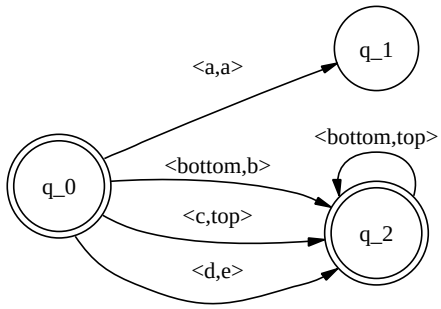

Figura 4.4: fs-autómata para complemento de un símbolo

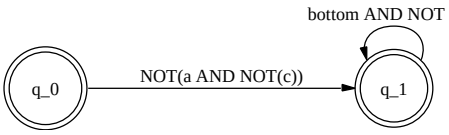

Figura 4.5: fs-pfsr para complemento de un símbolo

falso para los símbolos más particulares o más generales que a, y otro que es verdadero para todo símbolo del alfabeto.

Una tercer ventaja de los autómatas con predicados, es que permiten unificar todas las transiciones entre dos estados en una sola, que es la disyunción de los predicados correspondientes a ellas.

Los teoremas que siguen muestran que la definición de fs-autómata dada en la sección anterior, y su versión con predicados, efectivamente reconocen los mismos lenguajes.

Teorema 4.10 Dado un fs-autómata A, existe un fs-pfsr que reconoce el mismo lenguaje que A.

Para demostrarlo, sea $A = (Q, \Sigma, q_0, F, \delta)$ el fs-autómata. Se define el pfsr sobre estructuras atributo-valor $A' = (Q, \Sigma, \pi, E, q_0, F)$, donde E es el mínimo conjunto de transiciones que cumplen

- $(p, a \wedge \overline{c}, q) \in E$ sii $q \in \delta(p, a, b)$, siendo c la estructura más general que cumple $b \sqsubseteq c$, con b distinto de $\top$.
- $(p, a, q) \in E$ sii $q \in \delta(p, a, \top)$

Previamente, se demuestra el siguiente lema:

Lema 4.11 Existe un camino $\pi = (\langle p_i, a_i, b_i, q_i\rangle)_{i=1..n}$ en A, con $a_1 \dots a_n \sqsubseteq w$, y $w \sqsubseteq b_1 \dots b_n$, si y sólo si $(p_1, w, q_n) \in \hat{E}$.

Demostramos en un sentido aplicando el principio de inducción completa sobre el largo de π.

Si el largo de π es 0, el resultado se cumple trivialmente. Supuesto que la propiedad se cumple para todo camino π de largo menor que un cierto h mayor que 0, considérese el camino $\pi_h = (\langle p_i, a_i, q_i\rangle)_{i=1..h}$, con $a_1 \dots a_n \sqsubseteq w$ y $w \sqsubseteq b_1 \dots b_n$. Existe un camino $\pi_{h-1} = (\langle p_i, a_i, q_i\rangle)_{i=1..h-1}$, con $a_1 \dots a_{h-1} \sqsubseteq w'$ y $w' \sqsubseteq b_1 \dots b_{h-1}$, y $w = w'a$. Por la hipótesis inductiva, $(p_1, w', q_{h-1}) \in \hat{E}$. Ahora bien, como $q_h \in \delta(q_{h-1}, a_h, b_h)$, hay dos casos posibles:

1. $\langle q_{h-1}, a_h \wedge \overline{c}, q_h\rangle \in E$, con $b_h \neq \top$ y c la estructura más general de Σ que cumple $b_h \sqsubseteq c$. En ese caso, como $a_h \sqsubseteq a$ y $a \sqsubseteq b_h$, entonces el predicado $a_h \wedge \overline{c}$ se satisface para a, y por lo tanto $(p_1, w, q_h) \in \hat{E}$.

2. $\langle q_{h-1}, a_h, q_h\rangle$ y $b_h = \top$. Como $a_h \sqsubseteq a$, entonces el predicado a_h se satisface para a, y por lo tanto $(p_1, w, q_h) \in \hat{E}$.

En el sentido inverso, el lema se demuestra aplicando inducción completa sobre el largo de w (se asume para la demostración que el fs-pfsr no tiene transiciones épsilon).

Si el largo de w el lema se cumple trivialmente. Supuesto que la propiedad se cumple para todo w de largo menor que un cierto h mayor que 0, considérese la tira w tal que $\langle q_0, w, q\rangle \in \hat{E}$ y cuyo largo es h. Entonces, debe existir una tira w' tal que $\langle q_0, w', q'\rangle \in \hat{E}$ y $\langle q', \pi, q\rangle \in E$, con $w = w'd$ y π verdadero para el símbolo d. Por hipótesis inductiva, existe en A un camino $\pi_{h-1} = (\langle p_i, a_i, b_i, q_i\rangle)_{i=1..h-1}$, con $a_1 \dots a_{h-1} \sqsubseteq w'$, y $w' \sqsubseteq b_1 \dots b_{h-1}$. Además, por la definición de E, π sólo puede tener dos formas:

1. Si $\pi = a \wedge \overline{c}$ entonces debe cumplirse $a \sqsubseteq d$ y $d \sqsubseteq b$, siendo b el símbolo más particular que cumple $b \sqsubseteq c$. Por lo tanto, en A se cumple $q \in \delta(q', a, b)$, con $a \sqsubseteq d \sqsubseteq b$.

2. Si $\pi = a$ entonces debe cumplirse $a \sqsubseteq d$. Por lo tanto, en A se cumple $q \in \delta(q', a, \top)$, con $a \sqsubseteq d$.

Por lo tanto, siempre existirá en A un camino $\pi_h = (\langle p_i, a_i, b_i, q_i\rangle)_{i=1..h}$, con $a_1 \dots a_h \sqsubseteq w$, y $w \sqsubseteq b_1 \dots b_h$.

Si $w \in L(A)$, como existe un camino $\pi = (\langle p_i, a_i, q_i\rangle)_{i=1..n}$ en A, con $p_1 = q_0$, $q_h \in F$, y $a_1 \dots a_n \sqsubseteq w$, y por el lema anterior, $(q_0, w, q_h) \in \hat{E}$; por lo tanto $w \in L(A')$. Ànálogamente se muestra que el recíproco también se cumple.

Teorema 4.12 Dado un fs-pfsr A, existe un fs-autómata que reconoce el mismo lenguaje que A.

Como se mencionó en la sección 3.1.3, dado un fs-pfsr, como el alfabeto es finito y todos los predicados son verdaderos o falsos evaluados para cada uno de los símbolos del alfabeto, es posible construir un AFND que reconoce el mismo lenguaje. A su vez, en el teorema 4.6 se demostró que dado un AFND cualquiera cuyo alfabeto sean estructuras atributo-valor, existe un fs-autómata que reconoce el mismo lenguaje. Con ambos resultados, se demuestra el teorema.

Operaciones booleanas sobre predicados fs

Los fs-pfsr son un caso particular de pfsr y, por lo tanto, como se vio en la sección 3.1.3, todas las operaciones de determinización y minimización están definidas para estos autómatas. Para esto, deben definirse las operaciones de negación, conjunción y disyunción de predicados. Estas operaciones reciben predicados como argumentos y devuelven un predicado que, evaluado sobre un símbolo del alfabeto, toma, respectivamente, el mismo valor de verdad que la negación, conjunción y disyunción de los valores de verdad de sus argumentos, evaluados sobre el mismo símbolo. Los fs-predicados (en la forma en que han sido definidos), no son predicados con operaciones booleanas arbitrarias sino que siempre están en la llamada *forma normal disyuntiva* (es decir, son disyunciones de conjunciones de literales), con el objetivo de facilitar su manipulación. Por lo tanto, debe asegurarse que las operaciones de conjunción, disyunción y negación mantengan esa forma normal.

- Disyunción: dados dos predicados

$$\begin{aligned} \pi_1 &= c_1 \vee c_2 \vee \ldots \vee c_n \\ \pi_2 &= c'_1 \vee c'_2 \vee \ldots \vee c'_m \end{aligned}$$

  el predicado que satisface su disyunción se define de la siguiente forma:

$$\pi_1 \vee \pi_2 = c_1 \vee c_2 \vee \ldots \vee c_n \vee c'_1 \vee c'_2 \vee \ldots \vee c'_m$$

- Conjunción: dados dos predicados

$$\begin{aligned} \pi_1 &= c_1 \vee c_2 \vee \ldots \vee c_n \\ \pi_2 &= c'_1 \vee c'_2 \vee \ldots \vee c'_m \end{aligned}$$

  el predicado que satisface su conjunción se define de la siguiente forma:

$$\begin{aligned}\pi_1 \wedge \pi_2 &= (c_1 \vee c_2 \vee \ldots \vee c_n) \wedge (\vee c'_1 \vee c'_2 \vee \ldots \vee c'_m) \\ &= (c_1 \wedge c'_1) \vee (c_1 \wedge c'_2) \ldots \vee (c_1 \wedge c'_n) \vee \ldots \vee (c_n \wedge c'_n)\end{aligned}$$

- Negación: dado un predicado

$$\pi_1 = c_1 \vee c_2 \vee \ldots \vee c_n$$

y donde cada c_i es de la forma

$$c_i = a_{i1} \wedge a_{i2} \ldots \wedge a_{im_i}$$

el predicado que satisface su negación se define de la siguiente forma:

$$\begin{aligned}\overline{\pi_1} &= \overline{c_1} \wedge \overline{c_2} \wedge \ldots \wedge \overline{c_n} \\ &\quad \overline{a_{11} \wedge \ldots \wedge a_{1m_1}} \wedge \overline{a_{21} \wedge \ldots \wedge a_{2m_2}} \wedge \ldots \wedge \overline{a_{n1} \wedge a_{n2} \ldots \wedge a_{nm_n}} \\ &= (\overline{a_{11}} \vee \ldots \vee \overline{a_{1m_1}}) \wedge (\overline{a_{21}} \vee \ldots \vee \overline{a_{2m_2}}) \wedge \ldots \wedge (\overline{a_{n1}} \vee \ldots \vee \overline{a_{nm_n}}) \\ &= (\overline{a_{11}} \wedge \overline{a_{21}} \wedge \ldots \wedge \overline{a_{n1}}) \vee \ldots \vee (\overline{a_{1m_1}} \wedge \overline{a_{2m_2}} \wedge \ldots \wedge \overline{a_{nm_n}})\end{aligned}$$

Es muy sencillo probar que las definiciones anteriores devuelve predicados que satisfacen los valores de verdad de conjunción, disyunción y negación según la lógica proposicional, y que son fs-predicados, utilizando las propiedades de distribución de la conjunción frente a la disyunción, y las leyes de de Morgan.

De la definición de las operaciones, puede verse que el tamaño de los predicados (en lo que a cantidad de literales se refiere) crece combinatoriamente respecto al tamaño de los operandos. Por lo tanto, su cálculo puede parecer impracticable. Sin embargo, es posible detectar tautologías o contradicciones a partir de los operandos, basados en su estructura. Se volverá sobre esto cuando se vea la implementación de los predicados.

4.1.3. fs-transductores y fs-pfst

Así como se extendieron los reconocedores a fs-reconocedores, es posible extender la definición de transductores de estado finito a transductores sobre estructuras atributo-valor (fs-transductores).

Definición 4.13 (fs-transductor) *Un* transductor sobre estructuras atributo-valor (fs-transductor) *es una 5-tupla* $(Q, \Sigma_1, \Sigma_2, F, q_0, \delta)$ *donde* Q *es un conjunto finito de estados,* Σ_1 *y* Σ_2 *son alfabetos finitos de EAVs (donde se asume*

la existencia de un elemento $\perp$ que es más general que cualquier estructura del conjunto, y un elemento $\top$ que es más particular),F es un conjunto de estados finales, q_0 es un estado inicial y $\delta : Q \times (\Sigma_1 \times \Sigma_1)^\epsilon \times (\Sigma_2 \times \Sigma_2)^\epsilon \rightarrow 2^Q$ es una función de transición que cumple $q \in \delta(q, \epsilon, \epsilon)$.

La función δ puede extenderse para que opere sobre tiras. Sea $\hat{\delta} : Q \times \Sigma_1^* \times \Sigma_2^* \rightarrow 2^Q$, definida de la siguiente forma:

- $q \in \hat{\delta}(q, \epsilon, \epsilon)$
- Si $q_2 \in \hat{\delta}(q_1, w_1, w_2)$ y $q_3 \in \delta(q_2, \langle a_1, b_1\rangle, \langle a_2, b_2\rangle)$ entonces $q_3 \in \hat{\delta}(q_1, w_1a', w_2b')$ para todo a',b' que cumplen $a_1 \sqsubseteq a' \sqsubseteq b_1$ y $a_2 \sqsubseteq b' \sqsubseteq b_2$

La relación definida por un fs-transductor T será $R(T) = \{(w_1, w_2) \mid \hat{\delta}(q_0, w_1, w_2) \cap F \neq \emptyset\}$.

Para los fs-transductores, la definición de camino es similar a la dada para los fst:

Definición 4.14 (Camino en un fs-transductor) *Dado un fs-transductor $T = (Q, \Sigma_1, \Sigma_2, q_0, F, \delta)$, un* camino *de T es una secuencia $(\langle p_i, \langle a_{1i}, b_{1i}\rangle, \langle a_{2i}, b_{2i}\rangle, q_i\rangle)_{i=1..n}$ donde $q_i \in \delta(p_i, \langle a_{1i}, b_{1i}\rangle, \langle a_{2i}, b_{2i}\rangle)$ y $q_i = p_{i+1}$ para $i = 1..n-1$. Además, si $p_1 = q_0$ y $q_n \in F$, entonces el camino es* exitoso.

Sin embargo, en este caso cambia la relación entre camino exitoso y la función $\hat{\delta}$, ya que si $q_f \in \hat{\delta}(q_0, w_1, w_2)$, entonces existe un camino exitoso $(\langle p_i, \langle a_{1i}, b_{1i}\rangle, \langle a_{2i}, b_{2i}\rangle, q_i\rangle)_{i=1..n}$ tal que $a_{11} \ldots a_{1n} \sqsubseteq w_1 \sqsubseteq b_{11} \ldots b_{1n}$ y $a_{21} \ldots a_{2n} \sqsubseteq w_2 \sqsubseteq b_{21} \ldots b_{2n}$.

Los teoremas siguientes muestran que los fs-transductores reconocen las mismas relaciones regulares que los transductores de estado finito.

Teorema 4.15 Dado un fs-transductor $T = (Q, \Sigma_1, \Sigma_2, q_0, F, \delta)$, existe un transductor de estado finito $T' = (Q, \Sigma_1, \Sigma_2, q_0, F, \delta')$ que representa la misma relación que T.

Para demostrarlo, se define δ' de la siguiente forma:

- si $q \in \delta(p, \epsilon, \epsilon)$, entonces $q \in \delta'(p, \epsilon, \epsilon)$
- si $q \in \delta(p, \epsilon, \langle a_2, b_2\rangle)$, entonces $q \in \delta'(p, \epsilon, a')$, $\forall a' \in \Sigma_2$, con $a_2 \sqsubseteq a' \sqsubseteq b_2$.
- si $q \in \delta(p, \langle a_1, b_1\rangle, \epsilon)$, entonces $q \in \delta'(p, a', \epsilon)$, $\forall a' \in \Sigma_1$, con $a_1 \sqsubseteq a' \sqsubseteq b_1$.
- si $q \in \delta(p, \langle a_1, b_1\rangle, \langle a_2, b_2\rangle)$ entonces $q \in \delta'(p, a', b'), \forall a' \in \Sigma_1, b' \in \Sigma_2$, con $a_1 \sqsubseteq a' \sqsubseteq b_1$ y $a_2 \sqsubseteq b' \sqsubseteq b_2$

Y se demuestra previamente el siguiente lema:

Lema 4.16 Existe en T un camino $\pi = (\langle p_i, \langle a_{1i}, b_{1i}\rangle, \langle a_{2i}, b_{2i}\rangle, q_i\rangle)_{i=1..n}$ donde $a_{11} \dots a_{1n} \sqsubseteq w_1 \sqsubseteq b_{11} \dots b_{1n}$ y $a_{21} \dots a_{2n} \sqsubseteq w_2 \sqsubseteq b_{21} \dots b_{2n}$, si y sólo si existe en T' un camino $\pi' = (\langle p_i, a'_i, b'_i, q_i\rangle)_{i=1..n}$ con $a'_1 \dots a'_n = w_1$ y $b'_1 \dots b'_n = w_2$

La demostración de este lema es idéntica a la del lema 4.5, basada en el principio de inducción completa en el largo de los caminos, y utilizando la definición de δ'.

Demostrado el lema, es directo ver que si $\langle w_1, w_2\rangle \in R(T)$, existe en T un camino exitoso $\pi = (\langle p_i, \langle a_{1i}, b_{1i}\rangle, \langle a_{2i}, b_{2i}\rangle, q_i\rangle)_{i=1..n}$ donde $a_{11} \dots a_{1n} \sqsubseteq w_1 \sqsubseteq b_{11} \dots b_{1n}$ y $a_{21} \dots a_{2n} \sqsubseteq w_2 \sqsubseteq b_{21} \dots b_{2n}$, y, por el lema, existe un camino exitoso $\pi' = (\langle p_i, a'_i, b'_i, q_i\rangle)_{i=1..n}$ con $a'_1 \dots a'_n = w_1$ y $b'_1 \dots b'_n = w_2$. Por lo tanto, si $\langle w_1, w_2\rangle \in R(T)$, entonces $\langle w_1, w_2\rangle \in R(T')$. Análogamente, si $\langle w_1, w_2\rangle \in R(T')$, entonces $\langle w_1, w_2\rangle \in R(T)$.

El recíproco del teorema 4.15 también se cumple:

Teorema 4.17 Dado FST $T = (Q, \Sigma_1, \Sigma_2, q_0, F, \delta)$, existe un un fs-transductor $T' = (Q, \Sigma_1, \Sigma_2, q_0, F, \delta')$ que representa la misma relación que T.

La demostración es idéntica a la del teorema 4.6, que establecía la equivalencia entre autómatas finitos no deterministas y fs-autómatas. Dado A, se define δ' de la siguiente forma:

- si $q \in \delta(p, \epsilon, \epsilon)$, entonces $q \in \delta'(p, \epsilon, \epsilon)$
- si $q \in \delta(p, a, \epsilon)$ entonces $q \in \delta'(p, \langle a, a\rangle, \epsilon)$
- si $q \in \delta(p, \epsilon, b)$ entonces $q \in \delta'(p, \epsilon, \langle a, a\rangle)$
- si $q \in \delta(p, a, b)$, entonces $q \in \delta'(p, \langle a, a\rangle, \langle b, b\rangle)$.

Y de forma análoga al teorema mencionado, se prueba por inducción completa en los caminos de los autómatas, que $\langle w_1, w_2\rangle \in R(T)$, si y sólo si $\langle w_1, w_2\rangle \in R(T)$.

Así como los fs-autómatas pueden modelarse como pfsr que codifican en sus predicados la necesidad de una relación de subsumción entre los símbolos de una tira de entrada y las etiquetas de las transiciones, los fs-transductores pueden representarse naturalmente como pfst con predicados sobre estructuras atributo-valor

Definición 4.18 (fs-pfst) *Un* transductor de estado finito con predicados sobre estructuras atributo-valor (fs-pfst) *es un pfst* $(Q, \Sigma, \Pi, E, q_0, F)$ *cuyo alfabeto es un conjunto de EAVs y el conjunto de predicados* π *está definidos igual que en el caso de los pfsr.*

Teorema 4.19 Dado un fs-transductor $T = (Q, \Sigma_1, \Sigma_2, q_0, F, \delta)$, existe siempre un fs-pfst T' tal que $R(T) \equiv R(T')$, siendo $R(T)$ y $R(T')$ las relaciones definidas por T y T' respectivamente.

Si $(w_1, w_2) \in R(T)$, entonces habrá en el fs-transductor T un camino exitoso $\pi = (\langle p_i, \langle a_{1i}, b_{1i}\rangle, \langle a_{2i}, b_{2i}\rangle, q_i\rangle)_{i=1..n}$ donde $a_{11} \ldots a_{1n} \sqsubseteq w_1 \sqsubseteq b_{11} \ldots b_{1n}$. Sea $T' = (Q, \Sigma_1 \cup \Sigma_2, \Pi_1 \cup \Pi_2, E, q_0, F)$ un fs-pfst, donde E se define como el mínimo conjunto de transiciones que cumple

- $(p, \pi_1, \pi_2, q) \in E$ sii $q \in \delta(p, \langle a_1, b_1\rangle, \langle a_2, b_2\rangle)$
- $(p, \pi_1, \epsilon, q) \in E$ sii $q \in \delta(p, \langle a_1, b_1\rangle, \epsilon)$
- $(p, \epsilon, \pi_2) \in E$ sii $q \in \delta(p, \epsilon, \langle a_2, b_2\rangle)$
- $(p, \epsilon, \epsilon) \in E$ sii $q \in \delta(p, \epsilon, \epsilon)$

siendo $\pi_1 = a_1 \wedge \overline{c_1}$, siendo c_1 la estructura más general que cumple $b_1 \sqsubseteq c_1$, en caso de ser $b_1 \neq \top$, y $\pi_1 = a_1$ en caso contrario. Análogamente, $\pi_2 = a_2 \wedge \overline{c_2}$, siendo c_2 la estructura más general que cumple $b_2 \sqsubseteq c_2$, en caso de ser $b_2 \neq \top$, y $\pi_2 = a_2$ en caso contrario

Entonces, puede demostrarse, en forma análoga a como se hizo en el teorema 4.10, que las relaciones reconocidas por T y T' son la misma relación regular.

Teorema 4.20 Dado un fs-pfst T, existe un fs-transductor que reconoce la misma relación que T.

Dado un fs-pfst, como el alfabeto es finito y todos los predicados son verdaderos o falsos evaluados para cada uno de los símbolos del alfabeto, es posible construir un FST que reconoce la misma relación. A su vez, en el teorema 4.17 se demostró que dado un FST cualquiera cuyo alfabeto sean estructuras atributo-valor, existe un fs-transductor que reconoce el mismo lenguaje. Con ambos resultados, se demuestra el teorema.

4.1.4. Álgebra modificada de expresiones regulares

Los autómatas finitos y transductores están asociados a expresiones regulares. Existe un álgebra sobre lenguajes y relaciones regulares que premite representar este tipo de lenguajes, pudiéndose generar a partir de ellas, y utilizando operaciones definidas, autómatas que reconozcan los lenguajes que representan.

Como los fs-autómatas y fs-transductores, en la forma que se han definido, reconocen exactamente a los lenguajes y relaciones de los autómatas finitos y transductores, y como además existe una versión extendida de todos los

algoritmos que representan a los distintos operadores (basados ellos en las operaciones de unión, concatenación y clausura de Kleene, entre otras), la extensión del álgebra a fs-autómatas y fs-transductores es inmediata. En este trabajo, se utilizará como notación el álgebra de expresiones regulares de Xerox (véase el apéndice B), en su versión extendida a autómatas sobre estructuras atributo-valor.

4.2. Una aproximación inicial a la solución

Considérese la siguiente regla contextual:

$$\mathsf{noProp} \rightarrow \mathsf{uvTen}\ ^{*}(\mathsf{S}, 4)\ \backslash\ \mathsf{que}\ /\ \mathsf{uvInf};\ \mathsf{S} = \{\mathsf{uniVerb}, \mathsf{que}\}$$

Que podría leerse de la siguiente forma: "se agrega la etiqueta noProp al texto que va desde antes de la aparición del terminal que hasta después de él, si antes apareció en el texto el terminal uvTen, seguido de hasta cuatro terminales, los cuales no son ni uniVerb, ni que y luego aparece el terminal uvInf". Los diferentes componentes de su lado derecho puede escribirse como tres expresiones regulares (escritas en el álgebra de expresiones regulares dada en [4]):

- CIzq: uvTen [\uniVerb &\que]^ < 5
- Cuerpo: que
- CDer: uvInf

y por lo tanto existen tres autómatas finitos deterministas que los reconocen. La concatenación de estos autómatas reconoce el lado derecho de la regla.

Esta observación puede extenderse al lado derecho de todas las reglas contextuales consideradas. Esto se concluye de la forma del lado derecho, que es una secuencia de categorías (algunas de ellas opcionales o negadas) y zonas de exclusión definidas por enumeración.

Sea uno de los componentes del lado derecho de una regla (contexto izquierdo, cuerpo o contexto derecho), de la forma $c_1, c_2, \ldots, c_n$, donde los c_j son componentes de la forma cat, op(cat), no(cat), siendo cat una categoría, o pares (S, L) donde S es un conjunto de categorías y L es un natural. Entonces, existe una expresión regular (basada en el alfabeto dado por la enumeración de todas las categorías) que reconoce la secuencia, y está dada por: $e = e_1, e_2, \ldots, e_n$, donde

- si c_i es una categoría, entonces $e_i = c_i$
- si c_i es de la forma op(cat), siendo cat una categoría, entonces $e_i = (c_i)^+$

- si c_i es de la forma $\mathsf{no}(\mathsf{cat})$, siendo cat una categoría, entonces $e_i = \backslash c_i$
- si c_i es de la forma $(\{c_1, c_2, \ldots c_n\}, S)$ entonces $e_i = [\backslash c_1 \& \backslash c_2 \ldots \& \backslash c_n]\hat{} < S'$ siendo $S' = S + 1$.

Como e fue generada utilizando operadores de cálculo de estado finito, es efectivamente una expresión regular.

A partir de esa expresión, es posible generar un transductor que reconoce una tira compatible con la expresión regular asociada al cuerpo de la regla, y la devuelve seguida y precedida por marcas, siempre y cuando aparezca en el texto precedida de una tira reconocida por la expresión regular del contexto izquierdo y seguida de una tira reconocida por la expresión regular del contexto derecho. El transductor puede generarse a partir de una expresión de reemplazo con la siguiente forma genérica:

$$E(Cuerpo) \mathbin{@\!\!\rightarrow} \ \mathsf{m\_ini} \ \ldots \ \mathsf{m\_fin} \ \ || \ E(CIzq) \ \_ \ E(CDer)$$

donde E es la es la operación definida más arriba.

El modelado por reemplazos es posible porque en el lado derecho de la regla sólo hay terminales, y por lo tanto una regla no reescribe material previamente escrito por ella misma [12].

En el ejemplo utilizado más arriba, esto se transforma en la expresión regular

$$\mathsf{que} \mathbin{@\!\!\rightarrow} \ \mathsf{m\_ini} \ \ldots \ \mathsf{m\_fin} \ || \ \mathsf{uvTen} \ [\backslash \mathsf{uniVerb} \ \& \ \backslash \mathsf{que}]\hat{} < 5 \ \_ \ \mathsf{uvInf}$$

Si se considera al grafo de texto como un autómata finito F y se llama $T(R)$ al transductor generado a partir de una regla contextual R, entonces la siguiente expresión regular:

$$Id(F) \circ T(R)$$

indica la composición del autómata (visto como un transductor que computa la identidad) con el transductor que realiza el marcado. La proyección superior de esta expresión será un autómata que reconoce el texto original, con las marcas insertadas por la aplicación de la regla.

Se puede ver al grafo de texto como una estructura secuencial (considerando siempre como vértice siguiente al actual al alcanzado por la arista más larga o más corta que sale del vértice actual). Por tanto, por la propia semántica de las expresiones de reemplazo, el autómata resultante de la operación anterior también será una estructura secuencial.

A partir del autómata resultante, es posible modificar el grafo de texto original, agregando arcos entre las posiciones determinadas por las marcas, con información obtenida a partir de la propia regla a aplicar, como se verá más adelante.

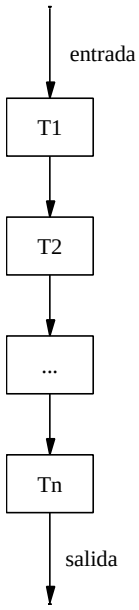

Figura 4.6: Cascada de Transductores

4.3. Aplicación simultánea de reglas

Hasta el momento, se ha supuesto que se aplica una sola regla al grafo de texto, y se esbozó una solución al problema. La aplicación del procedimiento descrito para todas las reglas de un módulo producirá un resultado equivalente a la aplicación del intérprete actual al grafo de texto. Sin embargo, al estar basada la solución en expresiones regulares de reemplazo, es posible utilizar una cascada de transducciones para aplicar simultáneamente todas las reglas de un módulo.

Una cascada de transductores es la composición de varios transductores [2], para obtener otro transductor que computa la misma transducción que la aplicación en secuencia de los operandos.

En este caso, es posible componer los transductores resultantes de la expresión correspondiente a cada regla para obtener un transductor que, compuesto con el grafo de texto visto como un autómata (de la misma forma que para el caso de una sola regla), agrega marcas resultantes de la aplicación al mismo tiempo de todas las reglas.

Si se tienen las siguientes reglas contextuales:

$$\begin{aligned} r_1 &\rightarrow CIzq_1 \setminus B_1 \,/\, CDer_1 \\ r_2 &\rightarrow CIzq_2 \setminus B_2 \,/\, CDer_2 \\ &\vdots \\ r_n &\rightarrow CIzq_n \setminus B_n \,/\, CDer_2 \end{aligned}$$

la siguiente expresión regular genera tal transductor:

$$\begin{aligned} E(B_1)\ @\rightarrow\ \mathsf{m\_ini_1}\ \ldots\ \mathsf{m\_fin_1}\ \ &||\ E(CIzq_1)\ \_\ E(CDer_1) \\ &.o. \\ E(B_2)\ @\rightarrow\ \mathsf{m\_ini_2}\ \ldots\ \mathsf{m\_fin_2}\ \ &||\ E(CIzq_2)\ \_\ E(CDer_2) \\ &.o. \\ &\vdots \\ &.o. \\ E(B_n)\ @\rightarrow\ \mathsf{m\_ini_n}\ \ldots\ \mathsf{m\_fin_n}\ \ &||\ E(CIzq_n)\ \_\ E(CDer_n) \end{aligned}$$

donde .o. es el operador de composición en el cálculo de expresiones regulares, y $\mathsf{m\_ini_k}$, $\mathsf{m\_fin_k}$ son símbolos especiales que denotan, respectivamente, el comienzo y fin del cuerpo de la regla $\mathsf{r_k}$.

Para ser exactos, esta forma de aplicación supone modificaciones a las expresiones regulares generadas a partir de las reglas. Estas modificaciones surgen del problema de que cuando una regla inserta marcas en el texto, esas marcas pueden alterar el contexto para la aplicación de reglas subsiguientes. Por lo tanto, será necesario ignorarlas tanto en las expresiones generadas por los contextos como en la correspondiente cuerpo de la regla. Considerando estas modificaciones, la expresión para el transductor de la composición de las reglas será:

$$\begin{aligned} E(B_1)/mr\ @\rightarrow\ \mathsf{m\_ini_1}\ \ldots\ \mathsf{m\_fin_1}\ \ &||\ E(CIzq_1)/mr\ \_\ E(CDer_1)/mr \\ &.o. \\ E(B_2)/mr\ @\rightarrow\ \mathsf{m\_ini_2}\ \ldots\ \mathsf{m\_fin_2}\ \ &||\ E(CIzq_2)/mr\ \_\ E(CDer_2)/mr \\ &.o. \\ &\vdots \\ &.o. \\ E(B_n)/mr\ @\rightarrow\ \mathsf{m\_ini_n}\ \ldots\ \mathsf{m\_fin_n}\ \ &||\ E(CIzq_n)/mr\ \_\ E(CDer_n)/mr \end{aligned}$$

donde / es el operador Ignore del cálculo de expresiones regulares y *mr* es una expresión regular que representa a todas las marcas pasibles de ser insertadas por las reglas a aplicar en simultáneo.

4.4. Determinización

En la solución propuesta hasta el momento, nada se ha supuesto sobre el determinismo de los transductores generados a partir de las expresiones de reemplazo respecto a su entrada, esto es, utilizando la terminología introducida en la sección 3.1.2, los transductores generados no tienen por qué ser subsecuenciales (deterministas respecto a su entrada). Esta característica no afecta la correctitud de la solución. Sin embargo, desde el punto de vista de la aplicación a una entrada dada, el hecho de contar con autómatas subsecuenciales es lo que permite garantizar que dicha aplicación pueda realizarse en orden lineal respecto al tamaño de la entrada, volviéndose esta característica cada vez más importante cuando los textos de entrada comienzan a crecer.

En la sección 3.1.2 se vio que no todos los transductores computan funciones subsecuenciales, y por lo tanto son determinizables. En la solución presentada, el uso del operador *Ignore* en las expresiones de reemplazo hace que los transductores generados a partir de ellas no sean secuencializables. Intuitivamente, esto se debe a que, dado que el número de marcas (resultantes de la aplicación de otras reglas) a ignorar es ilimitado, no es posible saber si, luego de la aparición de cierta cantidad de marcas, aparecerá en la entrada una cierta expresión. Considérese, por ejemplo, la expresión:

$$[\mathsf{a}, \mathsf{b}]/\mathsf{marca} \rightarrow \mathsf{m1} \ldots \mathsf{m2}$$

cuyo transductor puede verse en la figura 4.7. La idea para determinizar este transductor sería postergar la decisión de la emisión de la salida hasta el estado 4, donde se ha reconocido la expresión [a, b] en la entrada. Sin embargo, como en todos los estados puede aparece un número arbitrario de marcas, hay una cantidad infinita de caminos diferentes entre el estado 0 y el estado 4, y por lo tanto no es posible modelarlos en forma determinística. Una referencia completa del problema del determinismo en transductores puede verse en [21].

Para resolver este problema, se opta por poner un límite superior en el número de marcas que pueden aparecer entre dos símbolos en la entrada, y se introduce el operador $Ignore_n$, que implica ignorar hasta n apariciones de subtiras de una expresión dentro de una tira de otra.

Previamente, se define el operador $Intro_n(Expr)$ que, dado un lenguaje, devuelve el lenguaje formado por las mismas tiras, donde entre cada par de símbolos aparecen hasta n ocurrencias de la expresión $Expr$, y que está dado por la siguiente expresión regular:

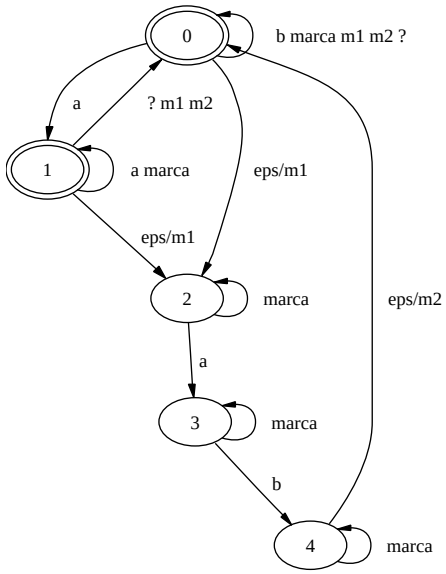

Figura 4.7: Transductor de Marcado con Ignore

$$Intro_n(Expr) : -[\underbrace{(\{\epsilon\} \times Expr) \ldots (\{\epsilon\} \times Expr)}_{nveces}\ id(?)]^*$$

El operador $Ignore_n(Expr_1, Expr_2)$ se define basado en $Intro_n$ de la siguiente forma:

$$Ignore_n(Expr_1, Expr_2) : -[Id(Expr_1)\ o\ Intro_n(Expr_2)].l$$

Una vez definido este operador, las expresiones de reemplazo son iguales a las definidas en la sección anterior, donde las apariciones del operador $Ignore$ son sustituidas por $Ignore_n$, con n fijo. Las expresiones de reemplazo así definidas generan transductores determinizables en todos los casos.

4.5. Reglas sobre estructuras atributo-valor

Hasta ahora se ha supuesto que en las reglas y en el grafo de texto (y por lo tanto en las expresiones regulares generadas) se trabaja sobre un alfabeto finito, donde está definida la operación de concatenación clásica.

En la extensión al enunciado original detallada en 2.1, tanto los arcos del texto como los ítems componentes de las reglas están etiquetados por un conjunto de parejas atributo-valor y la búsqueda en el grafo de texto se realiza no por identidad sino por subsumción. Por lo tanto, son una versión simple de las

estructuras atributo-valor descritas previamente (3.2), donde todos los valores asociados a los atributos son atómicos.

Habiendo sustituido el alfabeto de categorías atómicas utilizado hasta el momento por un conjunto F de estructuras atributo-valor, y utilizando las herramientas de estado finito (4.1) y el álgebra de expresiones regulares extendida a EAVs (4.1.4), es posible extender directamente la solución anterior para cubrir etiquetas compuestas.

4.6. Modificación del intérprete

El intérprete actual de reglas contextuales funciona aplicando las reglas por módulo. Esto es, se aplican todas las reglas de un módulo haciendo una recorrida del grafo de texto, para continuar luego con los siguientes. Para cada módulo de reglas, se ejecuta un componente de análisis del intérprete recibiendo como parámetro el tipo de vista del grafo y un indicador de si la aplicación de las reglas se realiza con o sin prioridad.

Lo que interesa es, entonces, sustituir en el intérprete el módulo de análisis actual (que utiliza una estrategia de *chart parsing*), por un nuevo módulo que aplique al grafo de texto la solución descrita en la sección anterior.

4.7. Diseño de la solución

La solución finalmente diseñada, se divide en dos partes: un módulo de generación que obtiene, a partir de un conjunto de reglas contextuales, las expresiones y transductores asociados, y un módulo de análisis del intérprete de reglas para el subconjunto de reglas contextuales considerado, que realiza la aplicación al grafo de texto de un grupo de ellas.

4.7.1. Módulo de generación

Este módulo recibe un conjunto de reglas y genera una cascada de transductores que permite el reconocimiento simultáneo de las reglas del conjunto.

A continuación se describen cada uno de los componentes del módulo:

Conjunto de reglas

Es un grupo de reglas contextuales a aplicar, con la característica de que en su lado derecho no aparece ninguna categoría que figure como lado izquierdo de otra regla del mismo conjunto.

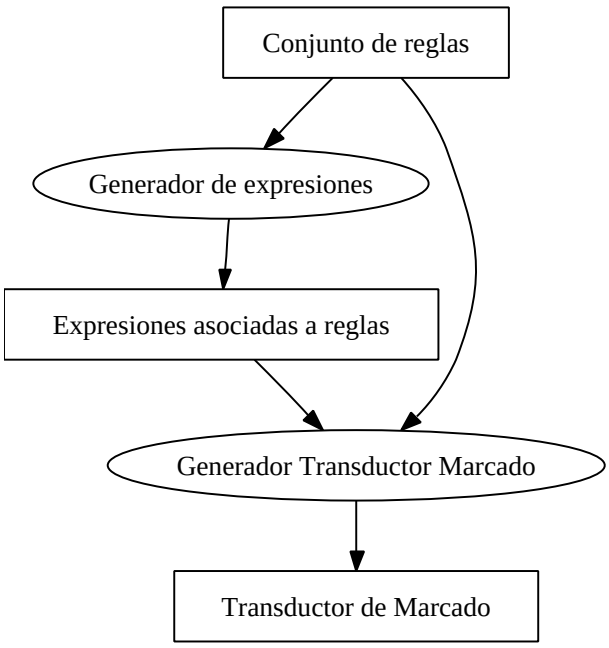

Figura 4.8: Módulo de Generación

Generador de Expresiones

Este componente es un proceso que, a partir del conjunto de reglas, genera las expresiones regulares de reemplazo asociadas a cada una de ellas, actuando en dos pasos principales

1. Se generan expresiones regulares extendidas a estructuras atributo-valor para el cuerpo, contexto izquierdo y contexto derecho de la regla, utilizando el algoritmo definido en 4.2.

2. Se generan expresiones de markup utilizando el álgebra de expresiones regulares extendida a estructuras, modelando las sustituciones, con dependencia de las prioridades indicadas, como se especificó en la sección 4.3. Las marcas insertadas son del tipo $[\mathsf{cat} : \mathsf{marca}, \mathsf{tipo} : \mathsf{inicio}, \mathsf{regla} : \mathsf{r}]$, y $[\mathsf{cat} : \mathsf{marca}, \mathsf{tipo} : \mathsf{fin}, \mathsf{regla} : \mathsf{r}]$. Es interesante notar en este punto que la expresión regular que denota "cualquier marca", puede expresarse sencillamente por la estructura $[\mathsf{cat} : \mathsf{marca}]$.

El resultado obtenido luego de este proceso es un conjunto de expresiones de reemplazo que insertan marcas antes y después de la ocurrencia del texto que da lugar a la aplicación exitosa de la regla.

Expresiones asociadas a reglas

Este componente es el resultado principal del módulo de generación, y consiste en un conjunto de expresiones regulares de reemplazo del tipo que se describió en 4.3.

Generador del Transductor de Marcado

Este proceso recorre las reglas contextuales de un conjunto y compila las expresiones de reemplazos asociadas a cada una, para finalmente componerlas, generando un único transductor que representa la aplicación en secuencia de los transductores asociados a cada una de las reglas.

El proceso involucra los siguientes pasos:

- Para cada regla se obtiene la expresión de reemplazo correspondiente a la aplicación de la misma.
- Se compila la expresión para obtener un transductor que recibe una secuencia de estructuras atributo-valor, y la devuelve con las marcas correspondientes a la aplicación de la regla. A este transductor se le aplica el algoritmo de determinización y se guarda para su posterior utilización.
- Una vez generados los transductores correspondientes a cada una de las reglas, se los lee y se los compone, de forma de generar un único transductor que representa la aplicación simultánea de todas las reglas. Nuevamente, se aplica el algoritmo de determinización al transductor obtenido. Este transductor, llamado *transductor de marcado* del módulo, se guarda para su posterior utilización por el módulo de aplicación

La generación de cada uno de los transductores de las reglas no es estrictamente necesaria, ya que para la aplicación sólo se utilizará el transductor de marcado para el módulo. Sin embargo, como este proceso es computacionalmente muy costoso, esto permite una generación incremental de los transductores de las reglas. Además, en caso de agregarse reglas al módulo, sólo se necesita generar los transductores asociados a las nuevas reglas, y recalcular la composición para obtener el nuevo transductor de marcado.

4.7.2. Módulo de aplicación

El módulo de aplicación permite, dado un grafo de texto etiquetado, agregar las aristas resultantes de la aplicación simultánea de una o más reglas contextuales.

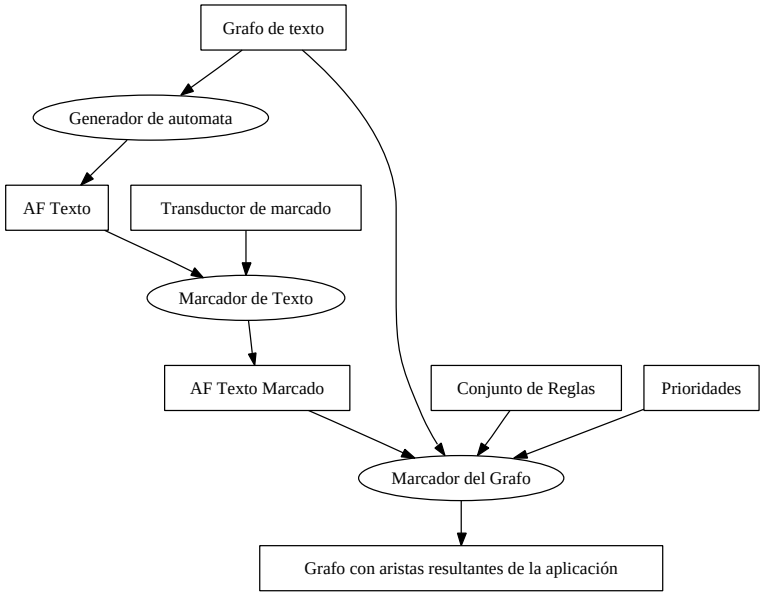

Figura 4.9: Módulo de Aplicación

Grafo de texto

Como se definió en 2.1, el grafo de texto es una estructura donde los vértices son las posiciones entre las palabras de cierto texto original, y las aristas están rotuladas con una etiqueta que indica el tipo de información que hay entre la posición inicial y final de la arista.

Al momento de aplicar las reglas, el grafo de texto tendrá dos tipos de aristas:

- Aristas iniciales: son las aristas que surgen a partir del etiquetado morfosintáctico que realiza un desambiguador/etiquetador. Están generalmente asociadas a descripciones relativamente estandarizadas de categorías morfosintácticas y sus rasgos típicos.
- Aristas funcionales: son aristas agregadas al grafo por el intérprete de las reglas contextuales

Las etiquetas del grafo de texto son conjuntos de parejas atributo-valor para ambos tipos de aristas.

Como ejemplo, en la figura 4.10 puede verse parte del grafo de texto generado a partir del texto: *"Cuando se editó un disco compacto de Ruben Rada en Buenos Aires no se consultó a Rada ni a sus colaboradores"*. La etiqueta entre los nodos 13 y 14 del grafo es resultado del etiquetado del desambiguador, mientras que la etiqueta entre las posiciones 13 y 16 surge de la aplicación del intérprete de reglas contextuales.

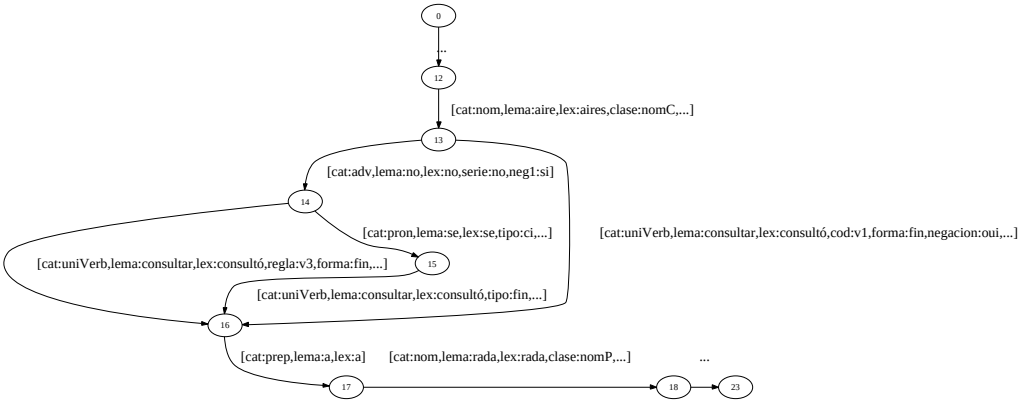

Figura 4.10: Grafo de texto

Generador de autómata de texto

Este componente permite transformar el grafo de texto en un fs-autómata (visto como un fs-pfsr).

La transformación se realiza de la siguiente forma: sea $GT = (V, A)$ el grafo de texto, siendo V los vértices (numerados secuencialmente de 1 en adelante), y A es un conjunto de aristas de la forma $(Ini, Fin, Etiqueta)$, donde Ini es la posición de inicio, Fin es la posición final, y $Etiqueta$ el valor del rótulo de la arista. Entonces, el autómata de texto AT será el fs-autómata $(\Sigma, Q, q_1, F, \delta)$, donde Σ es un alfabeto de estructuras atributo-valor Q un conjunto de estados donde $q_i \in Q$ si existe un vértice i en V, F es $\{q_n\}$, siendo n la cardinalidad de V, y $p_i \in \delta(Etiqueta)$ sii existe una arista $(j, i, Etiqueta)$ en A, y no hay ninguna etiqueta en A tal que $(j, i', Etiqueta')$ tal que $i' < i$.

Dado que, al realizar la transformación, sólo se tienen en cuenta para cada nodo aquella transición que llega al nodo más lejano, existirán nodos que, a pesar de tener transiciones de salida, no son destino de ninguna transición. Estos nodos son eliminados del autómata de texto, ya que no intervienen en el reconocimiento.

Por la forma en que se realiza la transformación, el autómata generado es una estructura secuencial, esto es, para cada estado sólo hay una transición de entrada y una de salida. Aunque los estados del autómata no son los mismos que los del grafo de texto es posible identificar cada transición del autómata con un arco del grafo de texto recorriendo secuencialmente el mismo a partir de su nodo inicial y tomando la transición máxima de salida de cada nodo.

En la figura 4.11 puede verse parte del autómata de texto generado a partir del grafo de texto construido en la sección anterior. Puede apreciarse que la numeración de los estados no tiene correspondencia ninguna con la numeración

de los nodos del grafo de texto.

Marcador de Texto

Dado el autómata de texto, el marcador de texto lo compone con el transductor de marcado para un módulo, y obtiene un transductor que tiene como lenguaje superior el texto original, y como lenguaje inferior el mismo texto con las marcas correspondientes a la aplicación de las reglas. A partir de este transductor, se proyecta su lenguaje inferior, para obtener el autómata de texto marcado que lo reconoce. Este autómata se utilizará posteriormente para insertar los arcos resultantes de la aplicación simultánea de las reglas al grafo de texto.

Considérese el autómata de texto de la figura 4.12. La composición con los transductores que reconocen las reglas:

$$
\begin{aligned}
&[\mathsf{cat} : \mathsf{iniProp}, \mathsf{lema} : \mathsf{por\ qué}, \mathsf{lex} : \mathsf{por\ qué}, \mathsf{cod} : \mathsf{pq2c}] \rightarrow \\
&\quad \backslash[\mathsf{cat} : \mathsf{prep}, \mathsf{lema} : \mathsf{por}][\mathsf{cat} : \mathsf{pronint}, \mathsf{lema} : \mathsf{qué}]/ \\
&[\mathsf{cat} : \mathsf{iniProp}, \mathsf{lema} : \mathsf{qué}, \mathsf{lex} : \mathsf{qué}, \mathsf{cod} : \mathsf{pq3}] \rightarrow \\
&\quad \backslash[\mathsf{cat} : \mathsf{prep}, \mathsf{lema} : \mathsf{por}]/
\end{aligned}
$$

devuelve el transductor de la figura 4.13, y su proyección de lenguaje inferior es el autómata finito de la figura 4.14.

Como el grafo de texto se considera una estructrua secuencial, esta aproximación es equivalente a la generación de una tira de EAVs a partir del grafo de texto, y la aplicación en cascada de los transductores de las reglas (o, de forma equivalente, del transductor de marcado). Esta equivalencia está dada porque el grafo de texto se considera siempre una estructura secuencial.

Prioridades

Es una base que modela una relación $Prior(r, \{r_1, \ldots, r_n\})$, que indica que si la regla r es exitosa, no debe aplicarse ninguna de las reglas $r_1, \ldots, r_n$.

Marcador del Grafo de Texto

Una vez generado el autómata de texto marcado, es necesario efectivizar la aplicación de las reglas sobre el grafo de texto. Para cada segmento del grafo de texto donde se identificó el lado derecho de una regla , se insertará un arco de acuerdo a lo indicado por el propio lado derecho de la regla en su estructura (las etiquetas inicio y fin).

En este momento del proceso se cuenta con un autómata de texto con marcas que preceden y siguen a la aparición en el texto del lado derecho de una regla. El algoritmo que agrega los arcos al grafo a partir de la información del autómata funciona de la siguiente forma:

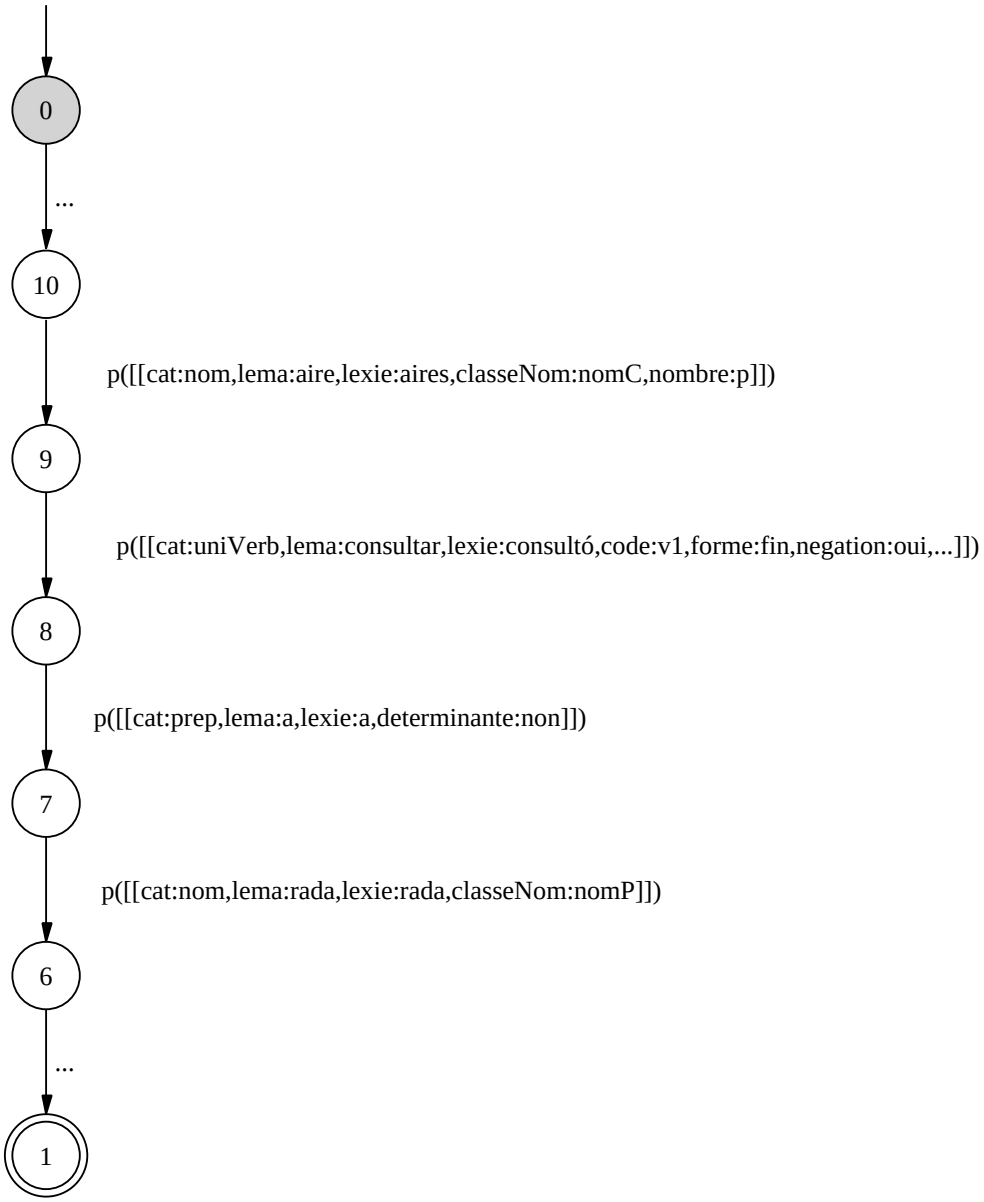

Figura 4.11: Autómata de texto

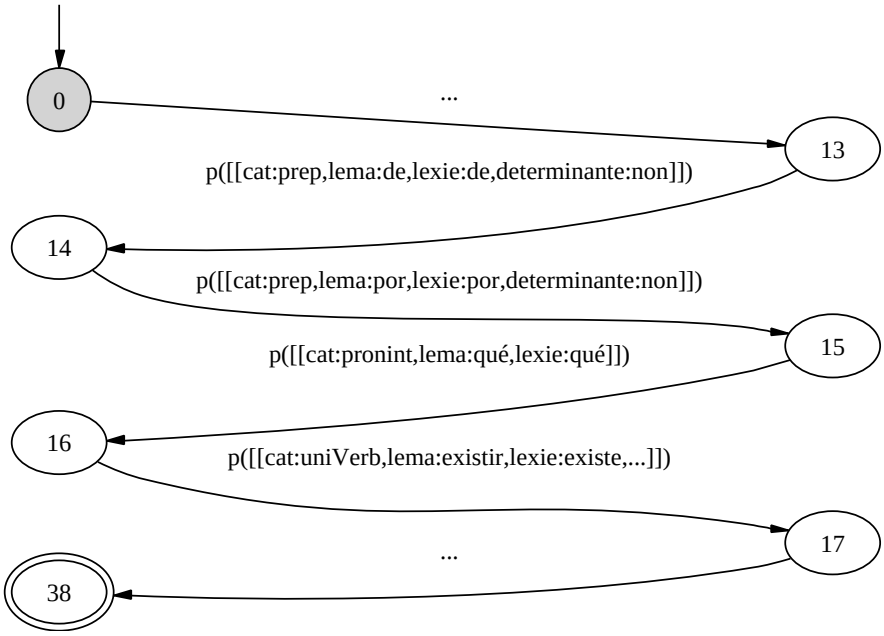

Figura 4.12: Autómata de texto original

- Partiendo del nodo inicial del grafo y del estado inicial del autómata, y mientras del estado del autómata no salga una transición etiquetada con una marca de inicio, avanza al estado siguiente y nodo siguiente de ambos.
- Cuando del estado actual sale una transición etiquetada con la marca de inicio de una regla, se recorre el autómata y el grafo al mismo tiempo, hasta encontrar el estado del que sale la marca de fin de la regla; con el nodo inicial y final, y con la etiqueta a insertar (que se obtiene de la propia regla), se agrega un arco temporal al grafo de texto. El proceso continúa en el nodo actual del grafo y en el estado siguiente al actual en el autómata.
- Cuando del estado actual sale una transición etiquetada con la marca de fin de una regla, el proceso continúa en el estado siguiente del autómata y en el nodo actual del grafo de texto.
- Finalizado el recorrido (cuando se llega al estado final del autómata y el nodo final del grafo de texto, lo que tiene que producirse al mismo tiempo), se transforman los arcos temporales en arcos del grafo de texto.

En el proceso se asume que en el autómata de texto sólo existe un estado

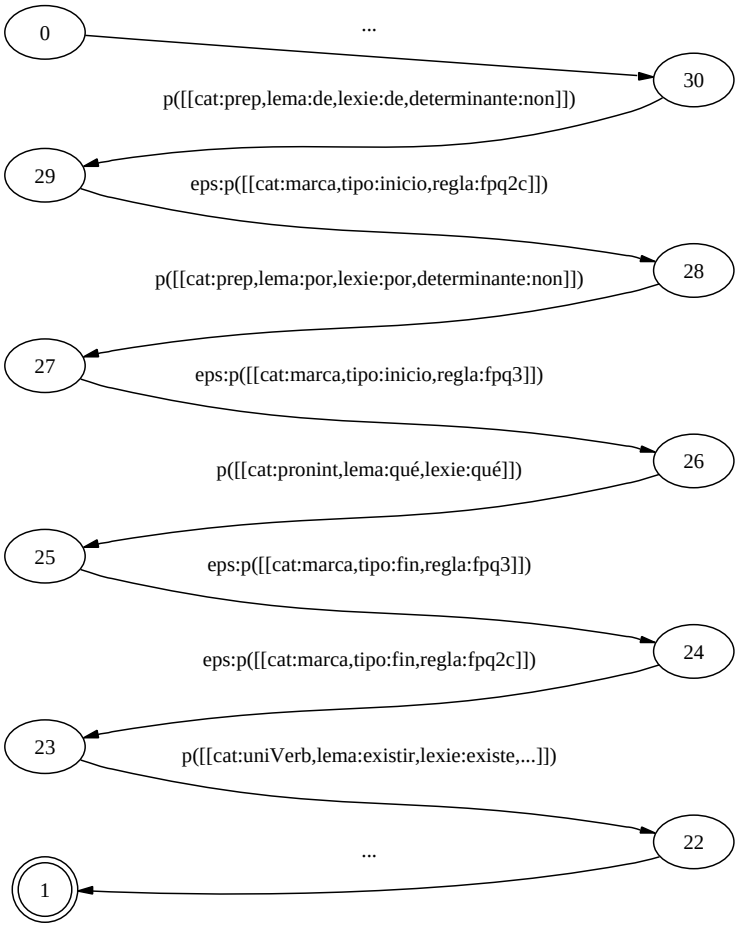

Figura 4.13: Transductor de texto marcado

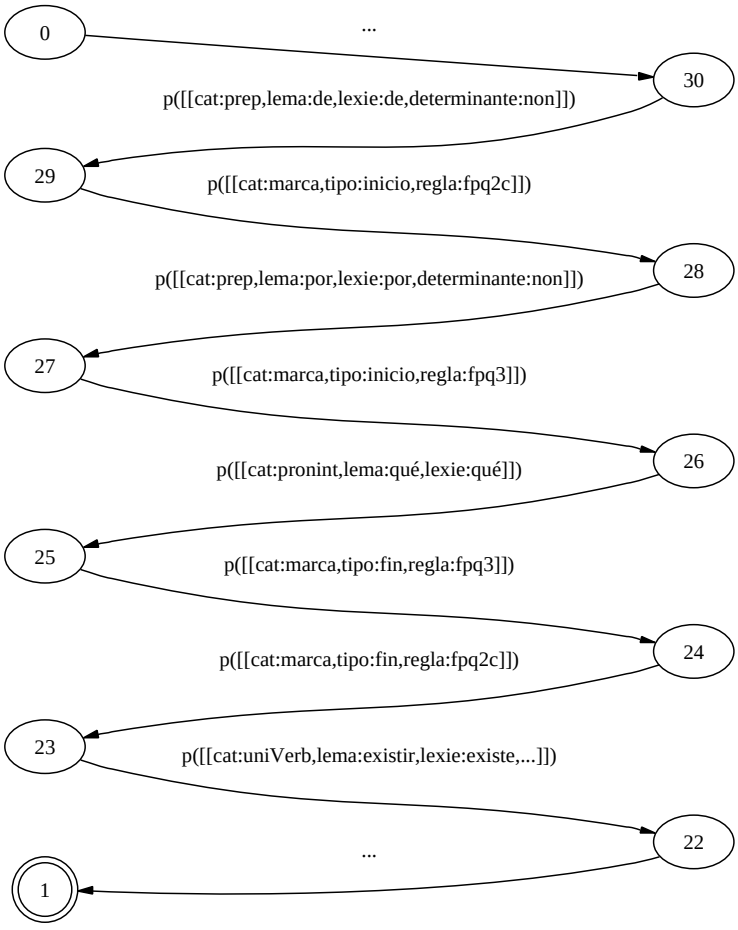

Figura 4.14: Autómata de texto marcado

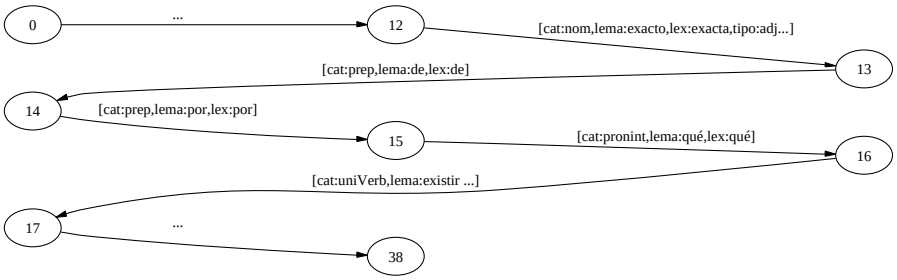

Figura 4.15: Grafo de Texto a Marcar

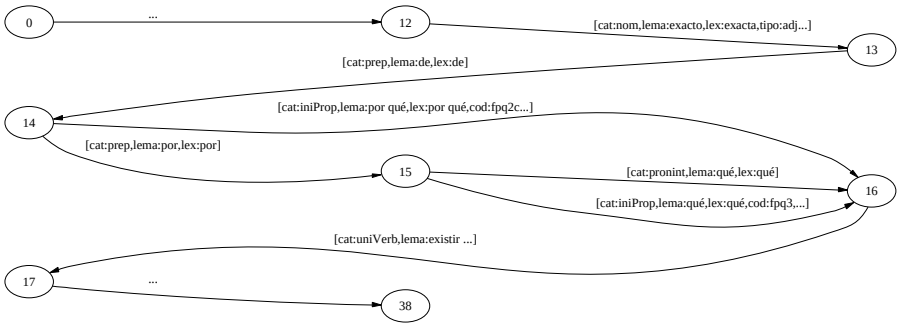

Figura 4.16: Grafo de Texto Marcado

siguiente al actual. Dada la forma en que se genera el autómata de texto (sección 4.7.2) y la forma de las expresiones de reemplazo (sección 4.7.1), esta propiedad siempre se cumple.

En el caso del grafo de texto sí puede haber más de un nodo siguiente al actual. Sin embargo, con idéntico criterio al utilizado al generar el autómata de texto, siempre se considera como nodo siguiente al actual al destino del arco máximo que sale del nodo actual. Esta identidad en los criterios es lo que permite sincronizar la recorrida de ambas estructuras.

Durante el proceso, los arcos resultantes de la aplicación de las reglas no son agregados directamente al grafo de texto, sino que se crean arcos temporales que son agregados al final del proceso. Esto se hace para evitar que los nuevos arcos afecten la recorrida del grafo y puedan hacer perder la sincronización con el grafo de texto.

Como ejemplo de este proceso, considérese la parte del grafo de texto que aparece en la figura 4.15, y el autómata de texto marcado generado a partir de él, que se consideró en la sección anterior. El proceso de marcado del grafo dejará al grafo como se muestra en la figura 4.16.

Aplicación de reglas con prioridad

En la sección 2.2, se mostró que el intérprete actual permite la aplicación de las reglas bajo un esquema de prioridades, que define que ciertas reglas no deben intentarse aplicar si la aplicación de otra regla fue exitosa, de acuerdo a la información contenida en la base citada en 4.7.2.

En el intérprete, el conjunto de reglas para las que se considera la prioridad está asociada a un cierto *head corner*, por lo que este sistema de prioridades está asociado más al funcionamiento del propio intérprete, y su estrategia de análisis, que a la especificación en sí del formalismo. Por lo tanto, la solución propuesta en este trabajo sólo podrá ser una aproximación a la existente, dado que la estrategia de aplicación es distinta, y no utiliza la información del *head corner* de las reglas.

Para resolver la aplicación con prioridad de un conjunto de reglas, la opción que se ha tomado es demorar la decisión de la aplicación efectiva de una regla al momento de la inserción del arco al grafo de texto. El proceso de marcado del grafo de texto calculará los arcos temporales (como se detalló en la sección anterior) correspondientes a la aplicación de *todas* las reglas, pero al momento de agregarlos al grafo, insertará solamente aquellos arcos que no estén cubiertos por un arco etiquetado con una regla de mayor prioridad; esto es, que no exista un arco que comience en la misma posición (o antes) y termine en la misma (o después), que el arco considerado.

En el ejemplo anterior, si la regla fpq2c tiene mayor prioridad que la regla fpq3, como el arco correspondiente a la primera regla va del nodo 15 al 16, y el de la segunda del nodo 14 al 16, entonces el primer arco no será agregado al texto, si la aplicación es con prioridad.

Capítulo 5

Implementación y Resultados

En este capítulo se muestran los detalles más relevantes de una implementación en Prolog de la solución planteada en el capítulo anterior. Previamente, debe especificarse la implementación de los autómatas sobre estructuras atributo-valor. En la sección 5.3 se muestran los resultados obtenidos al aplicar la solución a un cierto grupo de reglas contextuales.

5.1. Implementación de fs-autómatas

En esta sección se describe una implementación de fs-autómatas en Prolog. Como se especificó en la sección 4.1.2, se representarán como autómatas con predicados, donde estos últimos estarán definidos sobre estructuras atributo-valor relacionadas por subsumción y codificarán (además de agrupar los símbolos de las transiciones en clases) el comportamiento del autómata ante un símbolo de entrada. Se describe primero una representación en Prolog de las estructuras atributo-valor, para luego mostrar los predicados sobre estructuras atributo-valor. Finalmente, se integran estos predicados en una implementación de autómatas y transductores con predicados.

5.1.1. Estructuras atributo-valor

Para modelar las estructuras atributo-valor, se utilizará una versión modificada de la representación dada en [5], que codifica una estructura como una lista incompleta de parejas propiedad:valor.En la representación aquí utilizada se le agrega el functor *fs*, y se utilizan listas completas, modificando los predicados que computan la subsumción y unificación para que las considere como incompletas a los efectos de la comparación y unificación. En este trabajo, además, las propiedades tendrán valores atómicos y las estructuras no presentarán reentrancia. Por lo tanto, la EAV:

$$\begin{bmatrix} cat : verbo \\ persona : 3 \\ modo : indicativo \end{bmatrix}$$

se representará como fs([cat : verbo, persona : 3, modo : indicativo]).

Sobre esta representación, se definen además predicados Prolog que modelan las operaciones de subsumción y unificación.

5.1.2. Predicados sobre estructuras atributo-valor

Los fs-predicados se modelan como $p(Lista)$ done $Lista$ es una lista Prolog que representa la disyunción de sus elementos. En caso de que la lista tenga un solo elemento se abrevia $p([elem])$ como $p(elem)$.

Definición 5.1 (fs-predicados en Prolog) *Sea un conjunto F de estructuras atributo-valor, en el cual existe un elemento* [bottom] *más general que cualquier elemento del conjunto. Se definen literales, conjunciones y disyunciones:*

- *Literales: si $f \in F \cup \{[bottom]\}$ entonces f es un* literal positivo*, y $no(f)$ es un* literal negativo
- *Conjunciones: si L es una lista de literales, L es una* conjunción*.*
- *Disyunciones: si L es una lista de elementos, donde cada elemento es un literal o un predicado Prolog de la forma $and(ListAnd)$ siendo $ListAnd$ una conjunción, L es una* disyunción*.*

Entonces, un fs-predicado será un predicado Prolog de la forma $p(E)$, siendo E un literal, un predicado Prolog de la forma $and(C)$, siendo C una conjunción, o una disyunción.

Existe un predicado Prolog de evaluación $eval\_pred(+Pred, +Sym)$ (donde Sym es una estructura atributo-valor) que es verdadero si el fs-predicado $Pred$ se satisface para el símbolo de entrada. El valor de verdad para el predicado se determina de la siguiente forma:

- $eval\_pred(p(f), S)$ es verdadero si $f \sqsubseteq S$
- $eval\_pred(p(no(f), S)$ es verdadero si $p(f, S)$ es falso.
- $eval\_pred(p(and(Lista))$ es verdadero si para todo elemento $e \in Lista$, el predicado $eval\_pred(p(e), S)$ es verdadero.
- $eval\_pred(Lista)$ es verdadero si para algun elemento $e \in Lista$,el predicado $eval\_pred(p(e), S)$ es verdadero.

Construyendo la tabla de valores de verdad, puede verse que esta implementación es equivalente a la definición de predicados dada para los fs-pfsr.

5.1.3. Operaciones sobre predicados y simplificación

Las operaciones de conjunción, disyunción y negación son una implementación en Prolog de la definición dada para fs-pfsr, que garantizan devolver siempre como resultado fs-predicados.

Como ya se había mencionado, el problema que tiene la definición de las operaciones es que los predicados resultantes de estas operaciones crecen (en número de literales) en forma combinatoria respecto a la cantidad de literales de sus operandos. Los algoritmos de determinización y minimización hacen un uso intensivo de estas operaciones, por lo que los predicados resultantes (y el tiempo para calcularlos) podría volverse inmanejables. El objetivo de las reglas que siguen es la simplificación de los predicados obtenidos como resultado de las operaciones, a través de la detección de construcciones contradictorias, tautológicas o redundantes, utilizando la relación de subsumción entre las estructuras que componen el predicado, y la detección de estructuras incompatibles.

El funcionamiento general es el siguiente: dado un predicado, se simplifican primero cada una de las conjunciones componentes, para luego simplificar la disyunción obtenida. Todas las contradicciones obtenidas se representan por $\mathsf{p}(\mathsf{no}([\mathsf{bottom}]))$, y todas las tautologías por $\mathsf{p}([\mathsf{bottom}])$.

1. Si el fs-predicado $p(E)$ es una contradicción, entonces $p(and(C))$ es una contradicción para toda conjunción C, tal que $E \in C$. Por ejemplo,

   $$\mathsf{p}(\mathsf{and}([[\mathsf{cat} : \mathsf{verbo}], \mathsf{no}([\mathsf{bottom}])])$$

   es equivalente a

   $$\mathsf{p}(\mathsf{no}([\mathsf{bottom}]))$$

2. Si C' es el resultado de eliminar los elementos bottom de C, entonces $p(and(C))$ es equivalente a $p(and(C'))$. Por ejemplo,

   $$\mathsf{p}(\mathsf{and}([[\mathsf{cat} : \mathsf{verbo}], [\mathsf{bottom}], [\mathsf{persona} : 3]]))$$

   es equivalente a

   $$\mathsf{p}(\mathsf{and}([[\mathsf{cat} : \mathsf{verbo}], [\mathsf{persona} : 3]]))$$

3. Si $f \sqsubseteq f'$, $no(f) \in C$, $no(f') \in C$, entonces $p(and(C))$ es equivalente a $p(and(C'))$, siendo C' la lista resultante de eliminar los elementos f' de C. Por ejemplo,

   $$\mathsf{p}(\mathsf{and}([\mathsf{no}([\mathsf{cat} : \mathsf{verbo}]), \mathsf{no}([\mathsf{cat} : \mathsf{verbo}, \mathsf{modo} : \mathsf{indicativo}]), [\mathsf{persona} : 3]]))$$

   es equivalente a

   $$\mathsf{p}(\mathsf{and}([\mathsf{no}([\mathsf{cat} : \mathsf{verbo}]), [\mathsf{persona} : 3]]))$$

4. Si $f \not\sqsubseteq f'$, $f' \not\sqsubseteq f$, $no(f) \in C$, $no(f') \in C$, entonces $p(and(C))$ es equivalente a $p(and(C'))$, siendo C' la lista resultante de eliminar f y f' de C. Por ejemplo,

   $$\mathsf{p}(\mathsf{and}([\mathsf{no}([\mathsf{cat} : \mathsf{verbo}]), \mathsf{no}([\mathsf{cat} : \mathsf{nombre}]), [\mathsf{persona} : 3]]))$$

   es equivalente a

   $$\mathsf{p}(\mathsf{and}([[\mathsf{persona} : 3]]))$$

5. Si $f_1 \dots f_n \in F$, entonces $p(and(C))$ es equivalente a $p(and(C'))$, siendo C' la lista resultante de eliminar los elementos $f_1 \dots f_n$ de la lista, y agregar el elemento f resultante de su unificación. Si la unificación no está definida, entonces $p(and(C))$ es una contradicción. Por ejemplo,

   $$\mathsf{p}(\mathsf{and}([[\mathsf{cat} : \mathsf{verbo}], [\mathsf{persona} : 3], \mathsf{no}([\mathsf{cat} : \mathsf{verbo}, \mathsf{modo} : \mathsf{indicativo}])))$$

   es equivalente a

   $$\mathsf{p}(\mathsf{and}([[\mathsf{cat} : \mathsf{verbo}, \mathsf{persona} : 3], \mathsf{no}([\mathsf{cat} : \mathsf{verbo}, \mathsf{modo} : \mathsf{indicativo}])))$$

6. Si C es una conjunción de la forma $[e]$, entonces $p(and(C))$ es equivalente a $p(e)$. Por ejemplo,

   $$\mathsf{p}(\mathsf{and}([[\mathsf{cat} : \mathsf{verbo}]]))$$

   es equivalente a

   $$\mathsf{p}([\mathsf{cat} : \mathsf{verbo}])$$

7. Si $[\mathsf{bottom}] \in D$, entonces $p(D)$ es una tautología. Por ejemplo

   $$\mathsf{p}([[\mathsf{cat} : \mathsf{verbo}], [\mathsf{cat} : \mathsf{nombre}], [\mathsf{bottom}]])$$

   es equivalente a

   $$\mathsf{p}([\mathsf{bottom}])$$

8. Si $f \sqsubseteq f'$, $f \in D$, $f' \in D$, entonces $p(D) = p(D')$, siendo D' la lista resultante de eliminar f' de D. Por ejemplo

   $$\mathsf{p}([[\mathsf{cat} : \mathsf{verbo}], [\mathsf{cat} : \mathsf{verbo}, \mathsf{modo} : \mathsf{indicativo}], [\mathsf{cat} : \mathsf{nombre}]])$$

   es equivalente a

   $$\mathsf{p}([[\mathsf{cat} : \mathsf{verbo}], [\mathsf{cat} : \mathsf{nombre}]])$$

9. Si $f \sqsubseteq f'$, $no(f) \in D$, $no(f') \in D$, entonces $p(D)$ es equivalente a $p(D')$, siendo D' la lista resultante de eliminar $no(f)$ de D. Por ejemplo

   $$\mathsf{p}([\mathsf{no}([\mathsf{cat} : \mathsf{verbo}]), \mathsf{no}([\mathsf{cat} : \mathsf{verbo}, \mathsf{modo} : \mathsf{indicativo}]), [\mathsf{cat} : \mathsf{nombre}]])$$

   es equivalente a

   $$\mathsf{p}([\mathsf{no}([\mathsf{cat} : \mathsf{verbo}, \mathsf{modo} : \mathsf{indicativo}]), [\mathsf{cat} : \mathsf{nombre}]])$$

10. Si $f \in D$, $no(f) \in D$, entonces $p(D)$ es una tautología. Por ejemplo

$$\mathsf{p}([\mathsf{no}([\mathsf{cat} : \mathsf{verbo}]), [\mathsf{cat} : \mathsf{verbo}]), [\mathsf{cat} : \mathsf{nombre}]])$$

es equivalente a

$$\mathsf{p}([\mathsf{bottom}])$$

11. Si D es una disyunción de la forma $[e]$, entonces $p(D)$ es equivalente a $p(e)$. Por ejemplo

$$\mathsf{p}([[\mathsf{cat} : \mathsf{verbo}]])$$

es equivalente a

$$\mathsf{p}([\mathsf{cat} : \mathsf{verbo}])$$

5.1.4. Transductores fs-pfst

Una vez definida la implementación de estructuras atributo-valor y predicados, para el modelado de los pfsr y pfst se utiliza la biblioteca FSA Utilities [23]. Esta biblioteca proporciona un módulo para definir el comportamiento de los predicados que permite especificar su forma, así como las operaciones de conjunción, negación y disyunción, y, a partir de esto, define el funcionamiento de los diferentes algoritmos [24] que permiten generar y operar sobre los autómatas generados a partir de las expresiones regulares. Lo que se hizo en este trabajo fue precisamente definir un nuevo módulo para la representación,manipulación y simplificación de predicados sobre EAVs. Adicionalmente, fue necesario modificar las expresiones regulares que manejaba la biblioteca, dado que se asumía un alfabeto compuesto por átomos Prolog, característica que no cumplen las estructuras atributo-valor, en la forma como fueron implementadas. Por lo tanto, fue necesario modificar la biblioteca para que permita aceptar símbolos de la forma $fs(Lista)$ (por una descripción completa de los cambios a la biblioteca FSA Utilities, véase A).

5.2. Implementación de la solución

5.2.1. Generación de las estructuras

En la implementación del intérprete, las reglas contextuales están escritas en Prolog, con la siguiente estructura:

$$regla(HC, Modulo, Regla, EtIns, lDer(IzqHC, DerHC), Cond, Acc).$$

donde

- HC - Head Corner: corresponde a una etiqueta que dispara la aplicación de la regla en el intérprete actual. Para este trabajo, no será tenido en cuenta.
- Modulo: es el nómbre del módulo (conjunto de reglas) al que pertenece la regla
- Regla: es el nombre de la regla
- EtIns es la etiqueta que se agregará al grafo de texto en caso de que la aplicación de la regla sea exitosa.
- IzqHC es una lista de predicados de la forma $ob(Etiqueta)$, $op(Etiqueta)$, $no(Etiqueta)$, o $ex(ListaEtiquetas, Largo)$, que indican respectivamente la necesidad de la aparición obligatoria en el grafo de texto de una etiqueta, su aparición opcional, la aparición de una etiqueta que no subsuma a la etiqueta de la regla, o una zona de exclusión, de acuerdo a los criterios vistos en la sección 2.1. Estas etiquetas deben aparecer a la izquierda de la etiqueta HC. Además, en esta lista aparecen los átomos inicio y fin, que indican el comienzo y fin del cuerpo en el lado derecho (para separarlo de los contextos izquierdo y derecho).
- DerHC: predicados (idénticos a las anteriores) que aparecen a la derecha del $HeadCorner$.
- Cond: condición (especificada como predicado Prolog) que debe cumplirse para la aplicación de la regla
- Acc: es la acción (especificada como predicado Prolog) que se tomará, además de la inserción del arco en el grafo de texto, en caso de que la regla sea exitosa.

El siguiente es un ejemplo de regla:

```
% pq2a noProp --> uvHabNeg \por qué/ vInf
regla([_,qué|_],ipr,pq2a,[noProp,'por qué','por qué',pq2a|_],
      lDer([ob([pronint,qué|_]),ob([prep,por|_]),inicio,
      ob([uniVerb,haber,_,_,_,_,_,oui|_])],
           [fin,ob([uniVerb,_,_,_,inf|_])]
      )
     ).
```

Como puede verse, las categorías de las reglas contextuales se representan por n-uplas de valores donde parte de la información (las propiedades) es implícita y está dada por la posición del valor en la tupla. Para representar esta información por medio de estructuras atributo-valor como las utilizadas en este trabajo, es necesario explicitar esta relación.

Considérese la siguiente categoría:

$$[\_\ ,\_\ ,\_\ ,\ \mathsf{adj}\ \mid\ \_\ ]$$

Esta categoría modela (en el sistema Clatex) un nombre que tiene el valor adj para el atributo clase. En este caso, se asume que el valor de cat, que es siempre el atributo en la primera posición, es nombre. Sin embargo, la etiqueta podría considerarse como un verbo (el valor de cat) que tiene adj como valor del atributo modo.

Este tipo de categorías genera ambigüedad al ver la etiqueta como una estructura, ya que para cada valor posible del primer atributo se obtiene una estructura diferente, en el ejemplo dado las estructuras $\{\mathsf{cat} : \mathsf{nombre}, \mathsf{clase} : \mathsf{adj}\}$ y $\{\mathsf{cat} : \mathsf{verbo}, \mathsf{modo} : \mathsf{adj}\}$.

En el marco de este trabajo se optó por explicitar la relación entre categorías y posiciones en las etiquetas, con un doble objetivo: validar que una etiqueta esté correctamente formada de acuerdo a esta estructura y poder ver sin ambigüedad una categoría como estructura atributo-valor

Para representar esta relación se tiene la siguiente información:

- un conjunto de posiciones consecutivas $Pos \subseteq \mathbb{N}$
- un conjunto A de atributos posibles para las categorías
- un conjunto V de valores enumerados de atributos
- una función parcial $P : A \to 2^V$ de valores posibles para algunos atributos, asumiéndose que cada atributo a para el que $P(a)$ no está definida puede tomar cualquier valor, ya sea en el conjunto V o fuera de él.
- una función $D : A \times V \times Pos \to 2^{Pos \times A}$, con

  $$pos_1 < pos_2 \text{ si } \langle a_2, pos_2 \rangle \in D(\langle a_1, pos_1, v_1 \rangle)$$

  que indica los atributos determinados en posiciones siguientes a la actual, a partir del valor del atributo en la posición actual.

La función D, cumple la siguiente propiedad: si

$$\langle a_2, pos_2 \rangle \in D(\langle a_1, pos_1, v_1 \rangle)$$

y

$$\langle a_3, pos_3 \rangle \in D(\langle a_2, pos_2, v_2 \rangle)$$

entonces necesariamente no se cumple

$$\langle a, pos_3 \rangle \in D(\langle a_1, pos_1, v_1 \rangle)$$

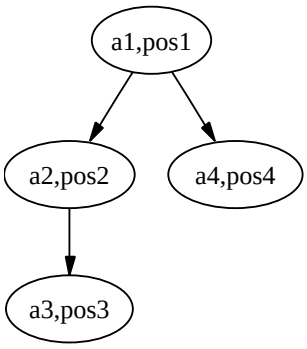

Figura 5.1: Grafo de la base lingüística

para ningún $a \in A$.
Además, si

$$\langle a_2, pos_2 \rangle \in D(\langle a_1, pos_1, v_1 \rangle)$$

entonces

$$\langle a, pos_2 \rangle \notin D(\langle a_1, pos_1, v_1 \rangle)$$

para todo $a \in A$, con $a \neq a_2$.

Por su definición, esta información puede representarse por medio de un grafo donde cada nodo es un atributo en una posición y las aristas están etiquetadas por los valores que hacen que un atributo determine a otro. La condición de intransitividad implica que sólo hay aristas desde un nodo a sus hijos. Además existe un solo nodo raíz etiquetado por la posición 1 y la categoría que corresponde a esa posición. Como ejemplo, se muestra parte de la estructura para el sistema Clatex.

Esta caracterización de las categorías permite convertir cualquier etiqueta correctamente definida en una estructura de parejas propiedad:valor, recorriendo el conjunto de posiciones y obteniendo en cada una el atributo y su valor, en caso de existir.

Esta base lingüística se implementó en Prolog utilizando *facts* para modelar los atributos y los valores posibles (en caso de existir) para cada uno de ellos, y un predicado

$$determina(+Pos, +ListaAtr, ?PosDet, ?Atr)$$

que modela la relación D, y que indica que en la posición $PosDet$ aparecerá el atributo Atr, si en la posición Pos aparece alguno de los atributos de la lista $ListaAtr$.

Para el acceso a la base, se implementaron predicados auxiliares que permite, por ejemplo, dado un atributo en una posición, devolver todos los atributos

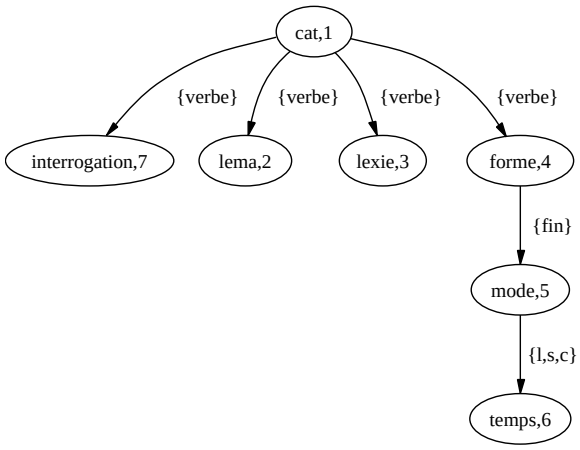

Figura 5.2: Base lingüística para Clatex

que éste determina directamente, realizando un recorrido en profundidad del grafo de atributos.

5.2.2. Módulo de generación

Como se explicó en el capítulo anterior, este módulo genera las expresiones de reemplazo a partir de las reglas contextuales.

La biblioteca de manipulación de autómatas de estado finito utilizada permite definir *macros* que modelan nuevos operadores a partir de los ya existentes. Teniendo en cuenta esta característica, el módulo generará un macro para cada regla contextual considerada, que tendrá el mismo nombre que la regla. Por lo tanto, el transductor de marcado asociado la regla cuyo nombre es fpq3, será generado por el macro del mismo nombre, a través del componente de generación del transductor de marcado.

Los macros asociados a cada regla se definen incrementalmente, basándose en una serie de macros generales, los cuales se detallan a continuación:

- *marcas_reglas*: esta expresión regular representa a cualquier estructura que represente una marca de regla. Es equivalente a fs([cat : marca]).

- *tags*: esta expresión regular representa a los tags introducidos por los algoritmos que implementan las expresiones de reemplazo. Como estos tags se incorporan al alfabeto durante el proceso, deben identificarse para ser eliminados luego de generado el transductor (ya que seguramente no aparecerán en la entrada del transductor).

- *elimino_tags(Expr)*: dada una expresión regular, genera un transductor

donde se han eliminado las apariciones de tags de marcado, como se especificó en el punto anterior.

- *tcompsigma(Expr)*: dada una expresión regular, identifica la aparición en la entrada de un símbolo *no* reconocido por la expresión, y que tampoco sea una marca ni un tag. Es equivalente a $term\_complement(\{Expr, marcas\_reglas, tags\}))$.

- *ignorar_marcas(Expr)*: reconoce una expresión regular, ignorando en la entrada la aparición de marcas. Inicialmente, la definición era equivalente a $Ignore(Expr, marcas)$, pero, como se explicó en la sección 4.4, se sustituyó por una versión de ignore que contempla hasta dos marcas entre cada par de símbolos de la tira de entrada. Esta versión está dada por la expresión $range(id(L)\ o\ introhasta2simbolos(Expr))$, siendo introhasta2simbolos la expresión $[option([] \times Expr), option([] \times Expr),\ id(?)]*$

- *markup_cx(Expr,MIni,MFin,CIzq,CDer)*: esta expresión implementa el operador de marcado con contexto, definido en 3.1.2. Reconoce la expresión regular $Expr$, y agrega antes y después de la aparición tiras correspondientes a los lenguajes dados por las expresiones $MIni$, $MFin$, si la aparición se produce en el contexto de tiras de los lenguajes $CIzq$ y $CDer$.

- *regsp(CIzq,Body,CDer,MIni,MFin)*: es la expresión a partir de la cual se generará el transductor de marcado para cada regla. Es equivalente a $markup\_cx(ignorar\_marcas(Body), MIni, MFin, CIzq, CDer))$.

Basados en estas macros, para generar la expresión asociada a una regla, se identifican los componentes de la regla y se generan las expresiones regulares para dichos componentes (recurriendo a la base para convertir las etiquetas a las correspondientes estructuras). Luego se genera el macro para la regla, definiéndolo como la aplicación del operador *regsp* definido anteriormente a las expresiones regulares que representan los componentes de la regla. La marca de comienzo de regla está dada por la expresión fs([cat : marca, tipo : inicio, regla : r], siendo r la regla que se está generando. La marca de fin es similar, indicando fin como valor de tipo.

La generación del transductor de marcado se realiza compilando las macros asociadas a cada regla, y componiéndolas para obtener un transductor único de marcado, como se explicó en el capítulo anterior.

5.2.3. Módulo de aplicación

Este módulo es una implementación en Prolog de la solución diseñada en la sección 4.7.2. Por razones de eficiencia, sin embargo, en la implementación se

sustituyó la composición del autómata de texto con el transductor de marcado por un proceso que simula la aplicación del segundo sobre el texto representado por el primero. Además, se fusionaron los procesos de generación del autómata de texto marcado y el marcado efectivo del grafo de texto original.

Generador de autómata de texto

Para generar el autómata de texto a partir del grafo, se implementó un procedimiento en Prolog que recorre secuencialmente el grafo de texto existente, teniendo en cuenta en cada caso la máxima arista que sale de cada nodo (en cado de existir más de una arista entre dos nodos, considera la última arista insertada por el intérprete), y genera un autómata secuencial como se definió en 4.7.2, convirtiendo una vez más las etiquetas a estructuras a través de la estructura asociada.

Marcador de Texto y del Grafo

Conceptualmente, este módulo realiza la composición entre el autómata de texto y el transductor de marcado, para obtener el autómata de texto marcado. Una primera opción fue realizar la composición entre ambos autómatas aplicado el algoritmo correspondiente para transductores. Sin embargo, al momento de la implementación, esta operación resultó muy costosa, a pesar de ser el transductor de marcado secuencial, debido a que el algoritmo debía calcular la conjunción entre el predicado de salida del estado del autómata de texto (visto, como se recordará, como un transductor que computa la identidad del lenguaje del autómata), y el predicado de entrada del correspondiente estado del autómata de marcado.

Por la semántica del proceso, la función del transductor de marcado siempre es devolver una secuencia igual a la de los símbolos de entrada, intercalados con símbolos correspondientes a marcas indicando la aplicación de las reglas. Por lo tanto, el proceso de marcado es equivalente a la aplicación del transductor a una entrada dada por la secuencia de predicados (que modelarán símbolos únicos) del texto. Basado en esta observación, se realizó una implementación en Prolog de un algoritmo que recorre el autómata de texto y, partiendo del estado inicial del transductor, evalúa los predicados de salida para el símbolo correspondiente del autómata de texto, generando las salidas correspondientes. Con la secuencia de símbolos de salida se genera luego el autómata de texto marcado.

En ese mismo proceso, para evitar una nueva recorrida sincronizada del autómata de texto marcado y del grafo de texto, se incorporó el agregado de los arcos al grafo de texto. El procedimiento, a medida que se mueve por las transiciones del autómata de texto, simultáneamente mantiene el nodo actual del grafo de texto; cuando aparece una marca de inicio en la salida, se guarda el nodo inicial y la regla, para luego, cuando se encuentra la marca

Regla	Tiempo(s)	Estados	Trans.	T.Comp.(s)	Estados	Trans.
pq1a	33	16	68	12	213	789
pq1b	31	16	68	855	95	464
pq1c	31	16	68	931	150	880
pq2a	101	69	442	1233	278	2580
pq2b	101	69	442	1211	353	3818
pq2c	33	16	68	1624	466	4674
pq3	12	7	23	1546	420	4312

Cuadro 5.1: Tiempos de generación de reglas

de fin correspondiente, agregar el arco temporal al grafo de texto. Finalmente, se transforman los arcos temporales en definitivos, teniendo en cuenta las prioridades, como se especificó en la sección 4.7.2.

5.3. Resultados obtenidos

A continuación se presentan algunos resultados obtenidos al aplicar la solución a un conjunto de reglas contextuales. Estos resultados apuntan a evaluar la implementación realizada, y se dividen en dos grupos principales: los correspondientes a la generación del autómata de marcado a partir de las reglas (producto del módulo de generación), y la aplicación del mismo a textos de diferente tamaño. Las reglas utilizadas fueron tomadas del sistema Clatex, y los textos son parte del corpus de evaluación del mismo sistema.

5.3.1. Generación del transductor de marcado

El cuadro 5.1 muestra los tiempos (en segundos) para la generación de algunas reglas, así como el número de estados y transiciones de los transductores correspondientes. También se lista el tiempo necesario para la composición en cascada de cada regla con las anteriores.[1]

Como esta etapa del proceso sólo debe ejecutarse una vez (cuando se han definidos las reglas necesarias), el tiempo en la generación de los autómatas no es crítico.

5.3.2. Aplicación de reglas al texto

En los cuadros 5.2 y 5.3 se muestran los tiempos en segundos para la aplicación de dos conjuntos de reglas a textos de diferentes tamaños. Se distingue

[1]Las pruebas fueron realizadas en un procesador Pentium©III de 800Mhz, con 384Mb de memoria RAM

Texto	Palabras	Gen. AF Texto (s)	Marcado(s)	Total(s)	Pal p/seg
p32tag.txt	478	0.39	2.35	2.74	174
p35tag.txt	716	0.57	3.85	4.42	162
p38tag.txt	1145	0.88	6.28	7.16	160
p30tag.txt	3074	2.92	23.25	26.18	117
p40tag.txt	3777	3.34	25.03	28.37	133
p10tag.txt	4444	3.83	34.15	37.98	117

Cuadro 5.2: Tiempos de aplicación (Transductor de 420 estados)

Texto	Palabras	Gen. AF Texto(s)	Marcado(s)	Total(s)	Pal p/seg
p32tag.txt	478	0.41	0.79	1.20	398
p35tag.txt	716	0.55	1.50	2.05	349
p38tag.txt	1145	0.89	2.29	3.18	360
p30tag.txt	3074	2.85	15.86	18.72	164
p40tag.txt	3777	3.49	17.15	20.64	183
p10tag.txt	4444	3.80	21.40	25.20	176

Cuadro 5.3: Tiempos de aplicación (Transductor de 23 estados)

el tiempo en la generación del autómata de texto y el del propio marcado del grafo de texto a partir del texto original y el transductor de marcado asociado a las reglas. Se muestra además la cantidad de palabras procesadas por segundo en el proceso. En el primero, la aplicación corresponde a un transductor de 420 estados y 4312 transiciones, y en el segundo, a uno de 23 estados y 212 transiciones.

Comparada con soluciones tradicionales basadas en transductores, como la presentada en [4], e incluso con el intérprete actual de reglas, los resultados no son tan buenos como podría esperarse, sobre todo si se tiene en cuenta que las reglas consideradas no son muchas, y su estructura es bastante sencilla. Existen razones para este comportamiento:

- La utilización del operador $Ignore_n$ (ignorar hasta n símbolos) para ignorar las marcas de otras reglas, implica agregar n transiciones al autómata correspondiente a la expresión, antes y después de cada transición que reconoce una categoría. Los resultados presentados corresponden a una implementación del operador $Ignore_2$. En el cuadro 5.4 se muestran los tiempos y tamaños de los transductores correspondientes a las mismas reglas, sin ignorar marcas.

- La utilización de predicados en lugar de símbolos en los autómatas hace

Regla	Tiempo(s)	Estados	Trans.	T.Comp.(s)	Estados	Trans.
pq1a	7	4	9	0.05	5	11
pq1b	7	4	9	0.19	6	19
pq1c	7	4	9	0.371	8	34
pq2a	15	7	26	3	16	123
pq2b	15	7	26	9	21	192
pq2c	7	4	9	9	22	189
pq3	3	3	5	15	23	213

Cuadro 5.4: Tiempos de generación de reglas (sin operador Ignore)

que los distintos algoritmos (en particular el de determinización) deban evaluar repetidamente operaciones booleanas entre los predicados. A pesar de que en la implementación se intentó optimizar estas operaciones, siempre serán más costosas que el símple cálculo de identidad entre símbolos que utilizan los algoritmos tradicionales.

- Los transductores en las FSA Utilities se representan por estructuras Prolog, donde, en particular, las transiciones son una lista ordenados según los criterios del lenguaje. Esto hace que algunas operaciones (por ejemplo, el obtener el estado siguiente al actual en un autómata) deban buscar secuencialmente en la lista, con las consecuencias en tiempo que ello implica. Esto hace que algunos algoritmos sean muy sensibles a la cantidad de transiciones de los transductores involucrados

Algunas sugerencias posibles para la mejora de la implementación se comentan en el capítulo 6.

Capítulo 6

Conclusiones

El objetivo de este trabajo fue utilizar técnicas de estado finito para aplicar un subconjunto de las reglas de un formalismo más expresivo de análisis sintáctico (el de las reglas contextuales). Si se restringe el formalismo a reglas donde el lado derecho está compuesto sólo por terminales, es posible ver estas reglas como expresiones regulares de reemplazo.

A lo largo de este documento se mostró que esta aproximación es realizable. Más generalmente, podría afirmarse que el trabajo muestra que es posible aproximarse al análisis sintáctico superficial de texto irrestricto a través de técnicas de estado finito. Los resultados obtenidos permiten extraer varias conclusiones que pueden servir como orientación para futuros trabajos en el área:

- La aplicación de los transductores como forma de cálculo de la imagen de una transducción puede realizarse eficientemente, sobre todo si los transductores utilizados son deterministas respecto a su entrada.

- La relación entre el lenguaje superior (que define el conjunto de tiras sobre la que se aplica la transducción), y el inferior (que define las tiras resultantes), puede expresarse de forma elegante y compacta a través del álgebra de expresiones regulares, en particular por las expresiones de reemplazo y marcado, en sus diferentes versiones.

- Los diferentes operadores del cálculo de expresiones regulares permiten modelar un número importante de relaciones diferentes, desde el producto cartesiano de lenguajes a sustituciones de un lenguaje por otro en un cierto contexto y de acuerdo a reglas que describen la dirección y el alcance del reemplazo.

- Dado que la transducción resultante de la aplicación en secuencia de varios transductores puede representarse a través de un único autómata utilizando la operación de composición, permite aislar las diferentes alteraciones que diferencian las tiras de entrada y salida (modelándolas como expresiones de reemplazo o codificándolas directamente), verificar

su correctitud de forma independiente, para finalmente combinarlas en una única transducción que la computa su aplicación a un texto).

Esta estrategia tiene, sin embargo, algunas limitaciones:

- Los transductores crecen rápidamente cuando las expresiones se vuelven más complejas. Si bien modelan en forma compacta la composición de relaciones similares (por ejemplo, reemplazos con contextos ligeramente diferentes), el transductor de la cascada puede llegar al peor caso en el que la cantidad de transiciones se acerca al producto entre la cantidad de transiciones de los autómatas involucrados en la operación, cuando se componen expresiones muy diferentes. A pesar de que la representación con predicados permite en general reducir drásticamente este número, éste sigue siendo relevante.

- El reconocimiento de estructuras repetitivas con una cota superior implica la introducción de un número importante de estados para poder "contar" las ocurrencias en la entrada. Podría pensarse en autómatas modificados con estructuras auxiliares (como una pila, o un contador asociado a las transiciones), con la obvia desventaja del alejamiento del paradigma de estado finito.

- A diferencia de los autómatas finitos, no todos los transductores son determinizables respecto a su entrada, por lo que algunas situaciones representables por relaciones regulares podrían no ser calculables de forma eficiente.

Resumiendo, los transductores parecen ser muy adecuados para el análisis local de múltiples expresiones; en el caso de expresiones que involucran gran cantidad de símbolos, el costo de su cálculo puede ser muy importante en términos de memoria, y los resultados muy sensibles a la implementación realizada. De todos modos, este costo no afectaría de forma importante a la aplicación de estas máquinas de estado finito a una cierta entrada, ya que (con una implementación eficiente de las mismas), el tiempo para la aplicación sería aproximadamente lineal al tamaño de tal entrada.

Autómatas sobre estructuras atributo-valor

Para la resolución de este problema en particular, fue necesario, además, definir nuevas máquinas de estado finito (los fs-autómatas) y un álgebra asociada a ellas, con el objetivo de modelar situaciones donde el alfabeto tiene como símbolos estructuras de parejas propiedad/valor, vinculadas por una relación de subsumción, y donde los conjuntos determinados por las expresiones regulares está dada por esa misma relación, en lugar de la aproximación clásica

de identidad entre símbolos. La prueba de que este tipo de autómatas no agrega expresividad al formalismo (más allá del uso de un alfabeto más complejo, lo que por otra parte podría realizarse con autómatas tradicionales), y que la extensión del álgebra de expresiones regulares es casi directa, permite llegar a un marco donde es posible modelar las relaciones de forma similar a lo realizado hasta el momento, con la ventaja de representar de forma más natural la existencia de una relación de generalidad entre los símbolos del alfabeto.

Este tipo de estructuras puede ser de utilidad para modelar situaciones similares, no sólo en el área del análisis sintáctico, sino como parte de otras aproximaciones al procesamiento del lenguaje natural, como podría ser el análisis morfológico (por ejemplo, las características morfológicas asociadas a un lema podrían verse como propiedades de una estructura, en lugar de como parte de una tira, como es la aproximación tradicional), o la representación de diccionarios.

Si bien las estructuras sobre las que se trabajó se consideraron con valores atómicos y sin reentracia, los fs-autómatas sobre se definieron de tal forma que sería posible trabajar sobre las versiones más generales de este tipo de estructuras.

La misma idea utilizada para la implementación de autómatas sobre estructuras atributo-valor podría utilizarse para modelar otro tipo de relaciones entre símbolos del alfabeto, con el objetivo de representar autómatas con criterios diferentes para el reconocimiento. En lugar de la relación de subsumción, podría, por ejemplo, utilizarse otra relación de orden parcial entre los símbolos, o relaciones entre conjuntos.

Implementación

La implementación de la solución (sobre todo en lo que se refiere a tiempos en la compilación de las expresiones regulares), no fue todo lo satisfactoria que se esperaba en la etapa de diseño. Como se mencionó al momento de mostrar los resultados, las causas fueron diversas, pero probablemente el factor más importante fue la implementación en Prolog de los autómatas, muy sensible por su diseño al tamaño de los transductores involucrados en las diferentes operaciones.

Algunas formas de mejorar esta implementación podrían ser:

- Realizar una implementación más eficiente de los transductores, optimizando en particular operaciones atómicas comunes, como la obtención de los estados siguientes a un estado dado, o, inversamente, de las transiciones que llegan al mismo.

- Implementar directamente los autómatas sobre estructuras: la implementación actual se basa en predicados para modelar la relación de subsumción; el cálculo de esta relación podría incorporarse directamente en el

algoritmo de reconocimiento y en los de manipulación de transductores. Tal aproximación parece, sin embargo, demasiado costosa comparada con los posibles beneficios que traería.

- Optimizar la representación y el cálculo de predicados sobre estructuras atributo-valor. La representación de las propias estructuras, así como los predicados sobre ellas podría representarse de forma más eficiente (por ejemplo, utilizando patrones de bits). Podría considerarse, asimismo, una modificación a los algoritmos de cálculo de operaciones booleanas sobre predicados, detectando, por ejemplo, casos particulares y comunes, donde el cálculo general puede sustituirse por una versión más simple (para citar un ejemplo, la conjunción de un predicado que reconoce una estructura consigo mismo devuelve siempre el propio predicado). En este trabajo se hicieron algunos intentos en ese sentido, pero la mejora en los tiempos no fue significativa.

Apéndice A

Modificaciones a las FSA Utilities

Las FSA Utilities [23] son una colección de utilidades escrita en Prolog, para la construcción, manipulación y aplicación de autómatas finitos. Es posible compilar diferentes tipos de autómatas (reconocedores, transductores, transductores con pesos en las transiciones), a partir de las expresiones regulares que los denotan. Esta biblioteca fue utilizada en este trabajo, como se detalla en el capítulo 5, por contar con características particulares adecuadas para el mismo,como las que se enumeran a continuación:

- Implementa autómatas y transductores basados en predicados, que fueron los utilizados para una representación de los fs-autómatas. De fundamental importancia fue la capacidad de definir nuevos predicados para los autómatas, además de los provistos por la propia biblioteca,que representan conjuntos de símbolos en la forma in(ListaOrdenada) para expresar pertenencia y not_in(ListaOrdenada), para indicar no pertenencia, donde los argumentos son listas ordenadas de átomos Prolog, y que son verdaderos (respectivamente, falsos) para un símbolo, si el símbolo pertenece (no pertenece) a la lista especficada en el predicado. Más adelante, en este mismo apéndice, se detallan las definiciones de predicados utilizada para el trabajo.
- Permite definir nuevos operadores (macros) a partir de los ya existentes en la biblioteca, o asociados a manipulaciones de autómatas que reciben como parámetros. Como ejemplo, un operador que devuelve la concatenación de el autómata representado por la expresión $Expr$ consigo mismo, podría modelarse a traves de una macro $Dup(Expr)$, definida como $[Expr, Expr]$. Esta característica fue utilizada en el trabajo para, por ejemplo, definir un operador que representara la generación de una expresión asociada a una regla contextual.
- Tiene integrados a la biblioteca (ya sea en el núcleo de la misma, o como macros ya definidos), los principales operadores del álgebra de expresiones regulares provenientes de diferentes fuentes, tales como los operadores

de reemplazo de Karttunnen, así como los algoritmos de secuencialización y minimización de transductores de Mohri. Si bien algunos de ellos se presentan en la biblioteca como en estado experimental, fueron utilizados con total éxito en el trabajo.

- Permite generar representaciones de los autómatas en formatos que pueden ser visualizados con herramientas populares de visualización de grafos, tales como *dot*.[1] . Esto resultó de suma utilidad para la visualización de los resultados obtenidos y la verificación de su correctitud.
- Exporta predicados Prolog que permiten el uso de la biblioteca como un módulo de Prolog, lo que permitió la integración directa de la solución al problema al intérprete actual de reglas contextuales

A.1. El módulo de predicados sobre estructuras atributo-valor

Para implementar los predicados sobre las estructuras atributo-valor, se agregó a la biblioteca un nuevo módulo de predicados, llamado fsa_preds_feat. Este módulo implementa, de acuerdo a una interfase definida en la biblioteca, los predicados necesarios, los que se resumen brevemente los principales:

- true(?Pred): el predicado verdadero para todos los símbolos. Se implementó como p([bottom])
- evaluate_predicate(+Pred, ?Symbol): evalúa si el predicado dado por *Pred* es verdadero para la estructura *Symbol*, de acuerdo a las reglas de evaluación definidas en la sección 4.1.2
- conjunction(+P0, +P1, ?P: define el predicado que representa la conjunción de los predicados que recibe como argumentos.
- disjunction(+P0, +P1, ?P: define el predicado que representa la disyunción de los predicados que recibe como argumentos.
- negation(+P0, ?P: define el predicado que representa la negación de los predicados que recibe como argumentos.

Adicionalmente, debieron implementarse en este módulo los procedimientos para la simplificación de predicados, detallados en la sección 5.1.3.

[1] Parte de la herramienta *graphviz* de los laboratorios *AT&T*

A.2. Otras modificaciones

Debido a que la biblioteca asume que los símbolos son siempre átomos Prolog, y la implementación de estructuras atributo-valor realizada en este trabajo es a través de listas, fue necesario modificar el núcleo de la biblioteca para permitir que acepte, como parte de una expresión regular, estructuras de la forma $fs([L])$, donde L es una lista de parejas propiedad:valor, que representa a una estructura atributo-valor.

Si bien la biblioteca incluye versiones de los operadores de reemplazo, no estaba implementado el operador de markup considerando el contexto, que fue utilizado para modelar las expresiones asociadas a las reglas, el que fue implementado.

Las macros asociadas a estos operadores (y a otros algoritmos, como el de determinación de si un transductor es funcional y secuencial) debieron ser modificadas levemente para utilizar símbolos auxiliares que fueran estructuras en lugar de símbolos atómicos.

Apéndice B

Álgebra de expresiones regulares

Como se indica en el capítulo 3, los autómatas de estado finito reconocen un subconjunto de los lenguajes, los regulares, mientras que los transductores codifican conjuntos de pares ordenados entre tiras de un alfabeto, las relaciones regulares. Nuevos lenguajes y relaciones pueden construirse a partir de ellos, utilizando operaciones sobre conjuntos (aunque algunas de ellas, como la intersección, sólo pueden aplicarse a lenguajes, para garantizar que el resultado también es regular).

Alternativamente, es posible representar estos lenguajes y relaciones utilizando expresiones regulares que los denotan. En este apéndice se describen los principales operadores del álgebra de expresiones regulares definido en [4], la cual, por otra parte, se utiliza como notación en este trabajo. En esa misma referencia puede encontrarse una definición completa de las operaciones, las cuales se muestran aquí sólo como referencia.

La implementación final de la solución fue realizada utilizando el algebra de expresiones regulares de las FSA Utilities, en la que algunos operadores no se llaman exactamente igual a la definida. En caso de diferencia entre ambas notaciones, se especificará el nombre del operador respectivo en las FSA Utilities. Por una referencia completa, véase [23].

Debe notarse que cada lenguaje puede verse también como la relación identidad, que mapea cada tira de entrada consigo mismo.

Expresiones atómicas

- el símbolo 0 (EPSILON) denota al lenguaje compuesto por la tira vacía. FSA Utilities: []
- el símbolo ? (ANY) denota al lenguaje de todas las tiras de un solo símbolo.
- cualquier símbolo a denota al lenguaje compuesto solamente por la tira a.

- una pareja de símbolos $a : b$ denota la relación compuesta solamente por el par [”a” : ”b”].

Opcionalidad

- la expresión (A) denota la unión del lenguaje o relación A con el lenguaje compuesto por la tira vacía. FSA Utilities: $option(A)$.

Iteración

- la expresión $A+$ denota la concatenación del lenguaje A consigo mismo una o más veces.
- la expresión $A*$ denota la concatenación del lenguaje A consigo mismo cero o más veces.

Complemento

- la expresión $\ A$ denota el complemento del lenguaje A. A debe ser un lenguaje, porque las relaciones. regulares no son cerradas bajo complemento.
- la expresión $\backslash A$ denota el lenguaje compuesto por las tiras de un sólo símbolo que no están en A. FSA Utilities: $term\_complement(A)$.

Concatenación

- la expresión $[AB]$ denota al lenguaje compuesto por una tira del lenguaje A concatenada con una tira de B. FSA Utilities:$[A, B]$.
- la expresión $[A]\hat{}n$ denota a la concatenación n-aria de A consigo mismo.
- la expresión $[A] < \hat{}n$ denota a la concatenación de A consigo mismo, n o menos veces.

Contenido

- la expresión $\$A$ denota al lenguaje obtenido concatenando al lenguaje universal $?^*$ como prefijo y sufijo de A.

Ignore

- la expresión $[A/B]$ denota al lenguaje o relación obtenido a partir de A, eliminando ocurrencias de B^* en las tiras de A. En las FSA Utilities el operador se aplica en la forma $Ignore(A, B)$.

Unión

- la expresión $[A|B]$ denota la unión de los lenguajes o relaciones A y B.FSA Utilities:$\{A, B\}$.

Intersección

- la expresión $[A\&B]$ denota la intersección de los lenguajes A y B. Esta operación está definida sólo para lenguajes, porque las relaciones regulares no son cerradas bajo intersección.

Producto Cartesiano

- la expresión $[A.x.B]$ denota la relación que vincula toda tira de A con cada una de las tiras de B. A es llamado *lenguaje superior* y B *lenguaje inferior* de la relación.

Proyección

- $A.u$ denota al lenguaje superior de la relación A. En las FSA Utilities se expresa por $domain(A)$.
- $A.l$ denota al lenguaje inferior de la relación A. FSA Utilities:$range(A)$.

Composición

- la expresión $[A.o.B]$ denota la composición de la realción A con la relación B. Si A o B denotan un lenguaje, entonces es visto como la relación identidad sobre ese lenguaje. FSA Utilities:$A\ o\ B$.

Restricción

- la expresión $[A => L\_R]$ denota al conjunto de tiras tales que cualquier tira en A aparece siempre inmediatmente precedida por una tira de L, y seguida de una tira de R.

Reemplazo incondicional

- la expresión $[A- > B]$ denota la relación donde en cada tira del lenguaje universal (el lenguaje superior) se asocia con todos las tiras que son idénticas a ella, excepto que cada instancia de A que ocurre como subtira de la original se representa por una subtira de B.
- la expresión $[A(- >)B]$ al reemplazo opcional, esto es, la unión de $[A- > B]$ con la relación identidad sobre A

Marcado

- la expresión $[A- > B...C]$ denota la relación donde en cada tira del lenguaje universal (el lenguaje superior) se asocia la tira original, excepto que cada instancia de A que ocurre como subtira es representada como una copia que tiene una tira de B como prefijo y una tira de C como sufijo. B y C pueden omitirse.

Reemplazo condicional

- la expresión $[A- > B \mid\mid L\_R]$ denota al lenguaje donde cada tira del lenguaje superior es inmediatamente precedida por una tira en el lenguaje superior de L e inmediatamente seguida por una tira del lenguaje superior de R.

- la expresión $[A- > B//L\_R]$ denota al lenguaje donde cada tira del lenguaje superior es inmediatamente precedida por una tira en el lenguaje superior de L y la tira de reemplazo del lenguaje inferior es inmediatamente seguida por una tira del lenguaje R.

- la expresión $[A- > B\backslash\backslash L\_R]$ denota al lenguaje donde cada tira reemplazada en el lenguaje superior es inmediatamente precedida por una tira de L y la tira reemplazante es inmediatamente seguida por una tira de R

- la expresión $[A- > B\backslash/L\_R]$ denota al lenguaje donde cada tira reemplazante del lenguaje inferior es inmediatamente precedidad por una tira de L e inmediatamente seguida por una tira de R

Reemplazo dirigido

Las siguientes expresiones denotan una relación similar a $[A- > B]$, excepto que las subtiras a ser reemplazadas son seleccionadas bajo el régimen especificado.

- la expresión $[A@- > B]$ denota al lenguaje donde las tiras de reemplazo son elegidas de izquierda a derecha. Si más de una tira candidata comienza en una posición dada, sólo se reemplaza la más larga.

- la expresión $[A- > @B]$ denota al lenguaje donde las tiras de reemplazo son elegidas de derecha a izquierda. Si más de una tira candidata comienza en una posición dada, sólo se reemplaza la más larga.

- la expresión $[A@ > B]$ denota al lenguaje donde las tiras de reemplazo son elegidas de izquierda a derecha. Si más de una tira candidata comienza en una posición dada, sólo se reemplaza la más corta.

- la expresión $[A > @B]$ denota al lenguaje donde las tiras de reemplazo son elegidas de derecha a izquierda.

Si más de una tira candidata comienza en una posición dada, sólo se reemplaza la más corta.

Cada una de estas expresiones puede ampliarse para agregar un contexto de reconocimiento, idéntico al expresado en las expresiones que denotan al reemplazo condicional.

Bibliografía

[1] S. Abney. Parsing by chunks, 1991.

[2] S. Abney. Partial parsing via finite-state cascades, 1996.

[3] S. Ait-Mokhtar and J. Chanod. Incremental finite state parsing, 1997.

[4] K. R. Beesley and L. Karttunen. *Finite State Morphology*. CSLI Publications, Cambridge, England, 2003.

[5] P. Blackburn and K. Striegnitz. Natural language processing techniques in prolog. http://www.coli.uni-sb.de/ kris/nlp-with-prolog/html/.

[6] B. Carpenter. *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge, England, 1992.

[7] B. Carpenter, C. Pollard, and A. Franz. The specification and implementation of constraint-based unification grammars. In *Proceedings of the Second International Workshop on Parsing Technologies*, 1991.

[8] D. Clemenceau. Finite-state morphology: Inflections and derivations in a single framework using dictionaries and rules. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*. MIT Press, 1996.

[9] S. Eilenberg. *Automata, Languages and Machines*. Academic Press, 1974.

[10] G. Grefenstette. Light parsing as finite state filtering, 1996.

[11] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[12] R. Kaplan and M. Kay. Regular models of phonological rule systems. In *Computational Linguistics, 20(3)*, pages 331–379, 1994.

[13] L. Karttunen. The replace operator. In *Meeting of the Association for Computational Linguistics*, pages 16–23, 1995.

[14] L. Karttunen. Directed replacement. In A. Joshi and M. Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 108–115, San Francisco, 1996. Morgan Kaufmann Publishers.

[15] L. Karttunen, R. Kaplan, and A. Zaenen. Two-level morphology with composition, 1992.

[16] M. Mohri. On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2(01):61–80, Mar. 1996.

[17] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

[18] M. Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234(1–2):177–201, 2000.

[19] C. Pollard and I. A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications, Chicago, Illinois, 1994.

[20] E. Roche and Y. Schabes. Deterministic part-of-speech taggin with finite-state transducers. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*. MIT Press, 1996.

[21] E. Roche and Y. Schabes. Introduction. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*. MIT Press, 1996.

[22] M. D. Silberztein. The lexical analysis of natural languages. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*. MIT Press, 1996.

[23] G. van Noord. FSA utilities: A toolbox to manipulate finite-state automata. In *Workshop on Implementing Automata*, pages 87–108, 1996.

[24] G. van Noord and D. Gerdemann. Finite state transducers with predicates and identities.

[25] D. Wonsever and J.-L. Minel. Contextual rules for text analysis. *Lecture Notes in Computer Science*, 2004:509–??, 2001.