UNIVERSIDAD DE BUENOS AIRES

Facultad de Ciencias Exactas y Naturales

Departamento de Matemática

Morfología de estados finitos: El modelo de morfología dos niveles de Koskenniemi y su aplicación al modelado de la morfosintaxis de dos lenguas aborígenes argentinas

Tesis de Licenciatura

Alumno: Andrés Osvaldo Porta

Directora: Dra.María Cristina Messineo

Codirectora: Dra. Susana Puddu

Resumen

La lingüística moderna ha sido fuertemente influenciada por la teoría de lenguajes formales que recorre toda la informática. La aplicación más famosa de la teoría de lenguajes formales a la linguística es la gramática transformacional de Chomsky [16]. La estrategia de Chomsky fue considerar diferentes tipos de lenguajes formales para ver si tenían la capacidad de modelar la sintaxis de lenguajes naturales. El tipo más restrictivo, el menos potente, fue denominado lenguaje de estados finitos o regular. Chomsky demostró que la sintaxis de lenguajes naturales no puede ser efectivamente modelada por medio de los autómatas que modelan los lenguajes regulares, por tanto rechazó estos lenguajes como una teoría sintáctica y propuso que la sintaxis debe ser modelada por lenguajes más poderosos. Esto, sin embargo, no implica que se deba asumir que lo mismo debe suceder a nivel de la morfofonología de los lenguajes naturales. Un modelado por lenguajes regulares de este nivel es especialemente atrayente desde el punto de vista computacional porque permite implementaciones simples y eficientes.

Johnson [32] y Kaplan y Kay [33] demostraron, independientemente, que los formalismos fonológicos generativos usuales para componentes no cíclicos de la morfosintaxis de las lenguas naturales tienen una potencia expresiva equivalente a la de traductores de estados finitos. La aplicación de este resultado a la lingüística computacional produjo el modelo de morfología de dos niveles (Koskenniemi [41]). El propósito de esta tesis es presentar y discutir las características del modelo de morfología de dos niveles (MDN) desde el punto de vista de la teoría de lenguajes formales y de la teoría de la complejidad. Asimismo, con el propósito de ilustrar la aplicación de la teoría de lenguajes formales a la lingüística, se elaboran dos gramáticas que modelan la morfología verbal de dos lenguas aborígenes habladas en la argentina: el Quichua santiagueño y el toba.

La tesis se organiza en 4 capítulos:

En el Capítulo 1, se demuestra, siguiendo a Kaplan y Kay [34], que las reglas

morfológicas del modelo de Sound Pattern of English [18] son expresables por medio de relaciones regulares y por tanto pueden ser modeladas por traductores finitos.

En el Capítulo 2, se introduce el modelo de morfología de dos niveles (MDN) siguiendo la formulación de Koskenniemi [41]. Se muestra como el resultado del Capítulo 1 se puede aplicar a este modelo y se describe brevemente KIMMO, la implementación más conocida de este modelo. Como un ejemplo de aplicación se da un parser sobre PC-KIMMO para las formas verbales del quichua santiagueño.

En el Capítulo 3, se analizan y contrastan los resultados preexistentes para la complejidad del modelo de morfología de dos niveles. Si bien existen gramáticas formales que describen la morfología de algunas lenguas en particular para las que se ha demostrado que el tiempo de generación es lineal en el tamaño del input [29], a nivel abstracto, Barton [5] y [6] prueba que tanto generación y el análisis (parsing) en KIMMO son NP hard. Se expone también la dicusión de Church y Koskenniemi [42] acerca de estos últimos resultados.

En el Capítulo 4 se muestran algunas limitaciones que tienen los modelos regulares usuales para modelar de manera natural ciertos fenómenos morfofonológicos presentes en muchos lenguajes naturales. Se ofrecen ejemplos de estos fenómenos en lenguas aborígenes sudamericanas. Se presenta la alternativa propuesta de Creider, Hankamerz y Wood [19] para modelar de manera mas adecuada la morfología de los lenguajes que presentan estos fenómenos. Esta propuesta utiliza lenguajes lineales libres de contexto o, como se demuestra, sus dispositivos aceptores, los autómatas de dos cabezales. Finalmente se da una gramática lineal que modela la morfosintaxis del toba.

Los aportes originales de esta tesis son por una parte la elaboración de una gramática regular que describe la morfosintaxis de las formas verbales finitas del quichua santiagueño por medio del sistema KIMMO y por otra la identificación de las gramáticas lineales libres de contexto como las pertinentes para describir aspectos de la morfosintaxis de una clase aparentemente muy amplia de lenguas americanas. Partes de está tesis han sido presentadas en ponencias en el Congreso de la SAL del 2010, en Mendoza, Argentina y sobre ella se ha elaborado un trabajo que ha sido aceptado para la ACL 2010 Conference, en Uppsala, Suecia.

Índice general

Capítulo 1

Los sistemas de reglas fonológicas de SPE como modelos regulares

1.1. Definiciones y resultados previos de la teoría de lenguajes formales. Propiedades de los lenguajes regulares y de los autómatas finitos

Se repasan en esta sección definiciones y resultados de la teoría elemental de lenguajes formales que serán utilizados en el transcurso de la tesis. Las demostraciones de los resultados se pueden consultar en [30] y [53]

Un alfabeto Σ es un cualquier conjunto finito de símbolos. Un string sobre un alfabeto Σ es cualquier secuencia finita de símbolos de Σ. Notaremos con λ al string vacío, la secuencia que no tiene símbolos.

Definición 1.1 .

Dado un alfabeto Σ , se define recursivamente Σ^*, el conjunto de cadenas sobre Σ por:

1. *Base: $\lambda \in \Sigma^*$.*
2. *Paso recursivo: Si $w \in \Sigma^*$ y $a \in \Sigma$ entonces $wa \in \Sigma^*$.*
3. *Clausura: $w \in \Sigma^*$ si y sólo si puede obtenerse a partir de λ en un número finito de aplicaciones del operador de 2.*

Definición 1.2 *Dado un alfabeto Σ , se define $\Sigma^+ = \Sigma^* - \{\lambda\}$.*

Definición 1.3 *Un lenguaje sobre un alfabeto Σ es un subconjunto de Σ^*.*

Definición 1.4 *El producto de los lenguajes X e Y, notado XY, es el lenguaje: $XY = \{uv | u \in X \text{ y } v \in Y\}$.*

Definición 1.5 *Sea Σ un alfabeto. Los conjuntos regulares sobre Σ se definen recursivamente por:*

1. *Base: $\emptyset$, $\{\lambda\}$ y $\{a\}$, para todo $a \in \Sigma$ son conjuntos regulares sobre Σ.*
2. *Paso recursivo: Sean X e Y conjuntos regulares sobre Σ. Entonces los conjuntos: $X \cup Y$, XY y X^* son conjuntos regulares sobre Σ.*
3. *Clausura: X es un conjunto regular sobre Σ si y sólo si puede ser obtenido de los elementos base por medio de un número finito de operaciones de 2.*

Definición 1.6 *Sea Σ un alfabeto. Las expresiones regulares sobre Σ se definen recursivamente por:*

1. *Base: $\emptyset$, λ y a, para todo $a \in \Sigma$, son expresiones regulares sobre Σ*
2. *Paso recursivo: Sean u e v expresiones regulares sobre Σ. Entonces las expresiones: $(u \cup v)$, (uv) y (u^*) son relaciones regulares sobre Σ.*
3. *Clausura: u es una expresión regular sobre Σ si y sólo si puede ser obtenido de los elementos base por medio de un número finito de operaciones de 2.*

Definición 1.7 *Se denomina gramática de tipo cero a una cuatrupla (V, Σ, P, S), donde:*

1. *V es un conjunto finito de variables.*
2. *Σ (el alfabeto) es un conjunto finito de símbolos terminales.*
3. *S es un elemento distinguido de V, llamado símbolo inicial.*
4. *P es un conjunto finito de reglas, además $V \cap \Sigma = \emptyset$.*

Una regla a menudo se denomina producción y es un elemento del conjunto $V \times (V \cup \Sigma)$. Una producción $[A, w]$ se escribe $A \to w$.

Una regla de la forma $A \to \lambda$ se denomina regla nula o λ. Las gramáticas se usan para generar cadenas bien formadas sobre un alfabeto dado. El paso fundamental en el proceso de generación consiste en la transformación de una cadena por una aplicación de una regla.

Definición 1.8 *Se notará por $L(G)$, el lenguaje generado por la gramática G, al conjunto de strings sobre Σ generado por las reglas de G.*

Definición 1.9 *Dada $G = (V, \Sigma, P, S)$, una gramática (de tipo 0), diremos que G es de tipo 1 o sensitiva al contexto si toda producción $\alpha \to \beta$ verifica $|\alpha| \leq |\beta|$.*

Definición 1.10 *Una gramática de tipo 2 o libre de contexto es una tupla (V, Σ, P, S) donde toda regla P es un elemento del conjunto $V \times (V \cup \Sigma)^*$. Un lenguaje generado por una gramática de este tipo se denomina libre de contexto o de tipo 2.*

Ejemplo 1.11 *Consideremos la gramática $G = (V, \Sigma, P, S)$ con $V = \{S, A, B\}$, $\Sigma = \{a, b\}$ y P dado por:*

$$\begin{array}{cc} S \to aB & a \to bAA \\ S \to bA & B \to b \\ A \to a & B \to bS \\ A \to aS & B \to aBB \end{array}$$

El lenguaje generado por esta gramática es el conjunto de strings sobre Σ con igual cantidad de símbolos a y b. Esto se prueba por inducción en la longitud de la palabra.

Definición 1.12 *Una gramática de tipo 3 o regular es una gramática en la que toda regla tiene alguna las formas:*

1. $A \to a$
2. $A \to aB$
3. $A \to \lambda$ *donde* $A, B \in V$ *y* $a \in \Sigma$.

Un lenguaje generado por una gramática de este tipo se denomina regular o de tipo 3.

Ejemplo 1.13 *El lenguaje a^+b^+ es generado por la gramática regular G sobre el alfabeto $\Sigma = \{a, b\}$:*

$$G : S \to aS|aA$$
$$A \to bA|b$$

Definición 1.14 *Autómata Finito Deterministico*

Un autómata finito determinístico (AFD) es una tupla $M = (Q, \Sigma, \delta, q_0, F)$, donde:

1. *Q es un conjunto finito de estados.*
2. *Σ es un conjunto finito llamado alfabeto.*
3. *q_0 es un estado distinguido llamado estado inicial .*
4. *F es un subconjunto de Q llamado el conjunto de estados finales.*
5. *δ es una función de $Q \times \Sigma$ en Q conocida con la función de transición.*

Un AFD puede ser pensado como una máquina consistente de cinco componentes: un registro interno simple, un conjunto de valores del registro, una cinta, un lector de la cinta y un set de instrucciones. Los estados del AFD representan el estado interno de la máquina y son usualmente notados por: $q_0, q_1, \ldots, q_n$. El registro de la máquina contiene uno de los estados como su valor. Al inicio de la computación el valor del registro es q_0, el estado inicial del AFD. El input es una secuencia finita de elementos del alfabeto Σ. La cinta almacena el input hasta que sea necesario en la computación. Esta cinta está dividida en celdas, cada una de las cuales es capaz de almacenar un elemento del alfabeto. Como el largo de una cadena de input no está acotado, la cinta debe ser de de longitud infinita. El input de una operación de una AFD es ubicado en un segmento inicial de la cinta. El cabezal de la cinta lee sólo una celda del input. El cuerpo de la máquina consiste del cabezal y del registro. La posición del cabezal debajo de la celda de la cinta que este siendo escaneada. El estado actual del autómata esta dado por el valor del registro. La configuración inicial de la computación de la cadena *baba* se representa en la figura 1.11:

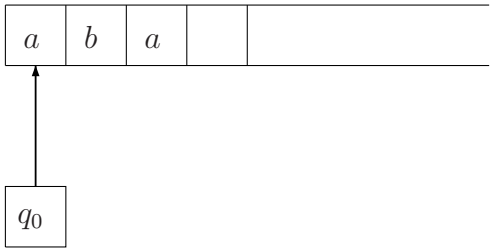


Figura 1.1: Configuración inicial de un autómata

La computación de un autómata consiste en la ejecución de una secuencia de instrucciones. La ejecución de una instrucción altera el estado de la máquina y mueve la cinta un lugar hacia la izquierda. El set de instrucciones se obtiene de la función de transición del AFD. El estado actual de la máquina y el símbolo escaneado por la máquina determinan la instrucción a ejecutar. La computación del autómata determina la aceptabilidad de un string. Se inicia el escaneo en estado q_0 leyendo el primer letra del string, luego se modifica (o no) el estado según el caracter escaneado y se mueve el cabezal un lugar hacia la derecha. Este procedimiento se continúa hasta llegar a un casillero en blanco. El string escaneado será aceptado si la computación termina en un estado final, de otra manera es rechazado.

Ejemplo 1.15 *Consideremos el autómata M definido por:*

$$
\begin{array}{lll}
M: & Q = \{q_0, q_1\} & \delta(q_0, a) = q_1 \\
 & \Sigma = \{a, b\} & \delta(q_0, b) = q_0 \\
 & F = \{q_1\} & \delta(q_1, a) = q_1 \\
 & & \delta(q_1, b) = q_0
\end{array}
$$

La secuencia de figuras a continuación representa los pasos de la computación por la cual es aceptado el string *aba*

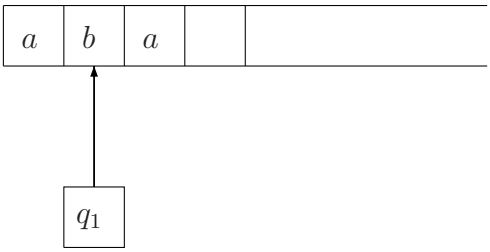


Figura 1.2: $\delta(q_0, a) = q_1$

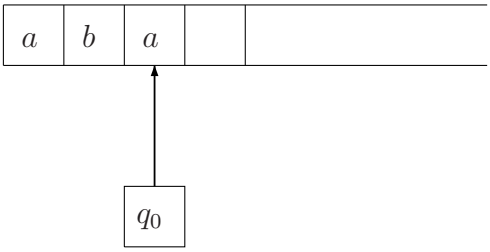


Figura 1.3: $\delta(q_1, b) = q_0$

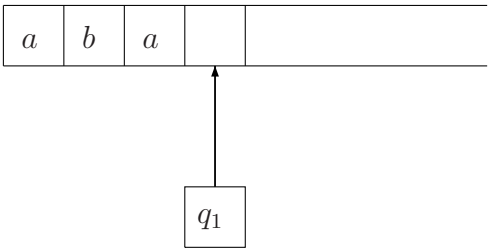


Figura 1.4: $\delta(q_0, a) = q_1$

Un AFD puede ser considerado un aceptor de lenguajes, el lenguaje reconocido por la máquina es el conjunto de cadenas que es aceptado por su operación. Dos máquinas con el mismo lenguaje aceptado se dicen equivalentes.

Definición 1.16 *Sea $M = (Q, \Sigma, \delta, q_0, F)$ un AFD. El lenguaje de M, notado $L(M)$, es el conjunto de strings en Σ^* aceptado por M.*

Ejemplo 1.17 *Definamos M un AFD que acepta el conjunto de strings sobre el alfabeto $\Sigma = \{a, b\}$ que contienen el substring bb(es decir $L(M) = (a \cup b)^* bb (a \cup b)^*$):*

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{a, b\}$

$F = q_2$

La función de transición δ se puede dar en una tabla de transiciones, en esta los estados se listan verticalmente y los símbolos del input horizontalmente. La acción del autómata en estado q_i con un input x se determina hallando la intersección de la fila y columna correspondientes. En este caso δ se define por:

δ	a	b
q_0.	q_0	q_1
q_1.	q_0	q_2
q_2 :	q_2	q_2

En estas tablas los estados finales se marcan con dos puntos.

Definición 1.18 *La función extendida de transición, δ^*, de un AFD con función de transición δ es una función de $Q \times \Sigma^*$ en Q definida por recursión en la longitud del string input por:*

1. *Base: Si $long(w) = 0$. Luego $w = \lambda$ y $\delta^*(q_i, \lambda) = q_i$. Si $long(w) = 1$, luego $w = a$, para algún $a \in \Sigma$ y $\delta^*(q_i, a) = \delta(q_i, a)$*

2. *Paso recursivo: Sea w un string de longitud n, $n > 1$. Luego $w = ua$ y $\delta^*(q_i, ua) = \delta(\delta^*(q_i, u), a)$*

Definición 1.19 *Diagrama de estados.*

El diagrama de estados para un AFD $M = (Q, \Sigma, \delta, q_0, F)$ es un grafo dirigido G definido por las siguientes condiciones:

1. *Los nodos de G son los elementos de Q.*

2. *Las marcas de los arcos de G son elementos de Σ.*

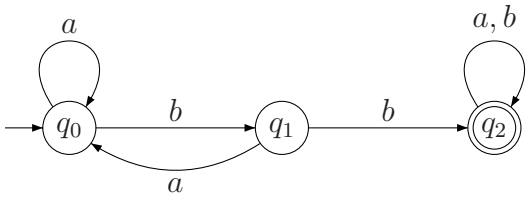
3. *q_0 es el nodo inicial que se marca con $>\bigcirc$.*

4. *F es el conjunto de nodos aceptores que se notan con círculo con doble contorno.*

5. *Un arco de un nodo q_i a otro q_j marcado con a nota una transición $\delta(q_i, a) = q_j$.*

6. *Para todo nodo q_i y simbolo a hay exactamente una arco marcado a y saliendo de q_i.*

Ejemplo 1.20 *El autómata del ejemplo anterior tiene el siguiente diagrama de estados:*



Teorema 1.21 *Sea $M = (Q, \Sigma, \delta, q_0, F)$ un AFD y sea $w \in \Sigma^*$. Luego w determina una único camino, p_w, en el diagrama de estados de M y $\delta^*(q_0, w) = q_w$.*

Teorema 1.22 *Sea $M = (Q, \Sigma, \delta, q_0, F)$ un AFD. Luego $M' = (Q, \Sigma, \delta, q_0, Q - F)$ es un AFD con $L(M') = \Sigma^* - L(M)$.*

Demostración Sea $w \in \Sigma^*$ y δ^* sea la función de transición extendida construída a partir de δ. Para cada $w \in L(M')$ se tiene:

$$w \in L(M') \Leftrightarrow \delta^*(q_0, w) \in Q - F \Leftrightarrow$$
$$\delta^*(q_0, w) \notin F \Leftrightarrow w \in \Sigma^* - L(M).$$

□

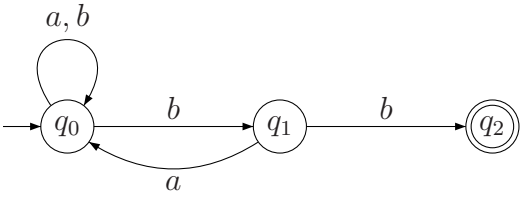
Definición 1.23 *Autómata Finito No Determinístico (AFND).*

Un autómata finito no determinístico es una tupla $M = (Q, \Sigma, \delta, q_0, F)$*, donde:*

1. Q *es un conjunto finito de estados.*
2. Σ *es un conjunto finito llamado alfabeto.*
3. q_0 *es un estado distinguido llamado estado inicial .*
4. F *es un subconjunto de* Q *llamado el conjuto de estados finales.*
5. δ *es una función de* $Q \times \Sigma$ *en* $\wp(Q)$ *conocida con la función de transición.*

El diagrama de estados para un AFND se define omitiendo la condición 6 en la definición del diagrama de estados de un AFD, por tanto puede existir más de un arco para cada símbolo desde un estado.

Ejemplo 1.24 *Se define* M *por:*



De la figura se ve claramente que:

$$L(M) = (a \cup b)^* bb$$

Teorema 1.25 *Los autómatas finitos no determinísticos y determinísticos son equivalentes.*

Teorema 1.26 *Un lenguaje regular* L *es aceptado por a AFD con alfabeto* Σ *si y solo si* L *es un conjunto regular sobre* Σ

Teorema 1.27 *Si L es un lenguaje regular L entonces L^R el lenguaje de los strings reversos de L es también regular.*

Demostración: Definamos primero el AFD M que es el aceptor de L. Dado M podemos construir ahora un AFND, M' equivalente a M con un solo estado final. En efecto, basta con elegir alguno de los estados finales de M como estado final de M'. Sobre el diagrama de estados de M, para todo otro arco que llegue a otro estado final de M, se construye otro arco marcado con el mismo símbolo que partiendo del estado original del arco en cuestión llegue al estado final seleccionado en la construcción de M'. Ahora se define M'' invirtiendo la dirección de los arcos del diagrama de estados de M' y tomando como inicial y final los estados final e inicial de M', respectivamente. Luego, por construcción: $L(M'') = L^R$ □

Lema 1.28 *Lema de Pumping para lenguajes regulares: Sea L un lenguaje regular que es aceptado por un AFD , M, con p estados. Sea s cualquier string en L con $|s| \geq p$. Entonces existen strings x, y, z tales que:*

1. $s = xyz$
2. $|y| \geq 1$ *y* $|xy| \leq p$
3. $xy^i z \in L, \forall i \geq 0$

Demostración Sea $M = (Q, \Sigma, \delta, q_0, F)$ el autómata que acepta L y $p = |Q|$ Supongamos que $s = a_1 a_2 \dots a_n \in L(M)$ con $n \geq p$. Se tiene entonces que:

$$\underbrace{q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots q_p}_{\substack{p+1 \\ estados}} \dots \xrightarrow{a_n} q_n \in F$$

Por el principio del palomar deben existir j y j', con $0 \leq j \leq j' \leq p$ tal que $q_j = q_{j'}$. Definamos ahora x, y y z reescribiendo la secuencia de arriba:

$$q_0 \xrightarrow{x}{}^{*} q_j \xrightarrow{y}{}^{*} q_{j'} (= q_j) \xrightarrow{z}{}^{*} q_n \in F$$

Luego $x = a_1 \dots a_j$, $y = a_{j+1} \dots a_{j'}$, $z = a_{j'+1} \dots a_n$ verifican, por definición, la proposición. □

Teorema 1.29 *Existe un lenguaje no regular.*

El lenguaje $L = \{a^i b^i, i \geq 0\}$ *no es regular.*

Demostración: Supongamos que L es regular y tomemos: $s = a^p b^p$ con p el valor del Lema de Pumping. Como $s \in L$ entonces por el Lema: $\exists x, y$ y z tales que:

1. $s = xyz$
2. $|y| \geq 1$ y $|xy| \leq p$
3. $xy^i z \in L, \forall i \geq 0$

Como $xyz = a^p a^p$, por el ítem 2, se tiene que:

$$x = a^k \text{ y } y = a^r, \text{ con } r > 0 \text{ y } k + r \geq p$$

Ahora como por 3., $xy^i z = xz \in L$. De allí se sigue la contradicción teniendo en cuenta que la cantidad de as se puede hacer arbitrariamente grande y la cantidad de bs es fija. □

Teorema 1.30 *Sean* L_1 *y* L_2 *dos lenguajes regulares. Entonces los lenguajes* $L_1 L_2$, $L_1 \cup L_2$ *y* L_1^* *son lenguajes regulares.*

Teorema 1.31 *Sean* L *es un lenguaje regular sobre* Σ *entonces el lenguaje* $\overline{L}$ *es regular.*

Demostración: Es inmediato considerando el autómata M aceptor de L y contruyendo a partir del teorema 1.22 el autómata M' aceptor de $\overline{L}$. □

Teorema 1.32 *Sean* L_1 *y* L_2 *dos lenguajes regulares. Entonces el lenguaje* $L_1 \cap L_2$ *es regular.*

Demostración Por la ley de De Morgan:

$$L_1 \cap L_2 = \overline{\left(\overline{L_1} \cup \overline{L_2}\right)}$$

y el miembro de la derecha es regular pues es construído a partir de L_1 y L_2 usando unión y complementación. □

1.2. Relaciones regulares y Traductores finitos

1.2.1. Definiciones y Resultados

Definición 1.33 *Una n-relación entre strings es una n-upla de* $(\Sigma^*)^n$*, que se notarán* $X =< x_1, \ldots, x_n >$.

En este contexto, utilizaremos λ para notar las n-relaciones con todas las coordenadas λ.

Definición 1.34 ***Operador de concatenación***

Dado un $n \geq 1$ *se define la n-concatenación en términos de la concatenación de los strings simples componentes de* $X =< x_1, \ldots, x_n >$ *e* $Y =< y_1, \ldots, y_n >$ *por:*

$$X.Y =: < x_1.y_1, \ldots, x_n.y_n >$$

La longitud de una n-relación X *se define como:*

$$|X| = \sum_{i=1}^{n} |x_i|$$

Observemos que $|\lambda| = 0$ y que $|X.Y| = |X| + |Y|$

Definición 1.35 *(n-Relación regular)*

Dado un alfabeto Σ*, si* $\Sigma^\lambda = \Sigma \cup \{\lambda\}$*, entonces son n-relaciones regulares:*

1. *El conjunto vacío y a, para todo* $a \in \Sigma^\lambda$.
2. *Si* R_1*,* R_2 *y* R *son relaciones regulares, entonces también lo son:*
   - a) $R_1.R_2$
   - b) $R_1 \cup R_2$
   - c) $R^* = \bigcup_{i=0}^{\infty} R^i$ *(clausura de Kleene)*
3. *no existen otras relaciones regulares*

Otras familias de relaciones puede ser también definidas por analogía con la caso de lenguajes formales. Por ejemplo un sistema de reglas libres de contexto puede ser usado para definir una gramática de n-relaciones libres de contexto introduciendo simplemente n-tuplas como los terminales de la gramática. El procedimiento de derivación libre de contexto produce árboles con n-concatenaciones en sus ramas. El presente análisis sobre sistemas de reglas fonológicas no necesita una potencia expresiva superior a la capacidad de las relaciones regulares y por eso nos concentraremos en estos sistemas más restringidos. Las relaciones que aquí se denominan regulares para enfatizar su conección con la teoría de lenguajes se denominan relaciones racionales en la literatura algebraica (Eilenberg [20]).

Definición 1.36 *Una expresión n-regular es una expresión cuyos términos son n-tuplas de símbolos alfabéticos o* λ.

Para simbolizar una n-tupla sus componentes se separarán por medio de dos puntos. Así, por ejemplo, la expresión $(a : b)\ (\lambda : c)$ describe la relación 2-regular que consiste del par $< a, bc >$, y $(a : b : c)^*\ (q : r : s)$ describe la 3-relación $\{< a^i q, b^i r, c^i s >, i \geq 0\}$. Esta notación permite escribir la operaciones de concatenación, unión y de clausura de Kleene.

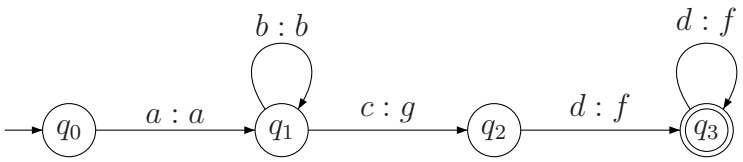
Definición 1.37 *Traductor Finito No Deterministico de n cintas(TFND).*

Un traductor finito no determinístico es una tupla $M = (Q, \Sigma, \delta, q_0, F)$*, donde:*

1. Q *es un conjunto finito de estados.*
2. Σ *es un conjunto finito llamado alfabeto.*
3. q_0 *es un estado distinguido llamado estado inicial.*
4. F *es un subconjunto de* Q *llamado el conjuto de estados finales.*
5. δ *es una función de* $Q \times \Sigma^\lambda \times \ldots \times \Sigma^\lambda$ *en* $\wp(Q)$ *conocida con la función de transición.*

En los análisis que siguen se trabajará con traductores de dos cintas. Los traductores se pueden representar, al igual que los autómatas, con diagramas de estados marcando los arcos con las n-tuplas del input.

Ejemplo 1.38 *El traductor M dado por:*



es el aceptor de la relación regular $(a:a)(b:b)^*(c:g)(d:f)^+$

Las propiedades que definen relaciones regulares, expresiones regulares y máquinas de estado finitos son la base un teorema bien establecido en correspondencia de Kleene que muestra la equivalencia de esta tres caracterizaciones de conjuntos de strings. La demostración de estas equivalencias es análoga a la de lenguajes y autómatas dadas por Hopcroft y Ullman[30].

Teorema 1.39 *Extensión del teorema de Kleene a relaciones regulares*

1. *Toda n-expresión regular describe una n-relación regular.*
2. *Toda relación n-regular es descripta por un expresión n-regular.*
3. *Todo autómata de n cintas acepta una relación n-regular.*
4. *Toda n-relación regular es aceptada por un traductor finito de n cintas.*

1.2.2. Propiedades de la relaciones regulares

Existen un número de conexiones básicas entre relaciones regulares y lenguajes regulares. Dada una relación R, se escribirá xRy si el par $< x, y >$ pertenece a la relación R. La imagen de un string bajo una relación R, x/R, es el conjunto de strings y tal que $< x, y >$ pertenece a R. Analogamente R/y es el conjunto de strings x tales que $< x, y >$ esta en R. Esta notación se extender a conjuntos de strings de manera obvia:

$$X/R = \bigcup_{x \in X} x/R$$

Esta convención notacional es útil para describir el uso del traductor como generador de strings en el dominio y en la imagen.

Teorema 1.40 *Son lenguajes regulares el dominio y rango de una relación regular R:*

$$Dom\,(R) = R/\Sigma^*$$

$$Rango\,(R) = \Sigma^*/R.$$

Demostración: En efecto, el rango y el dominio de la relación R son aceptados por las máquinas de estados finitos derivados del traductor $T\,(R)$ cambiando todas las transiciones $a : b$ a a y b respectivamente, para todo a y b en Σ^λ. □

Observación 1.41 *Dado un lenguaje L, la relación identidad, Id(L), que manda todo miembro de L en si mismo es regular.*

En efecto esta relación se puede definir por el traductor finito obtenido del autómata de estados finitos $M\,(L)$ cambiando las transiciones a por $a : a$.

Observación 1.42 *Claramente, para todos los lenguajes L:*

$$L = Dom\,(Id\,(L))$$

$$L = Rango\,(Id\,(L))$$

Observación 1.43 *El inverso, R^{-1}, de una relación regular R es regular pues es aceptado por un traductor formado por T(R) intercambiando todos los $a : b$ por $b : a$.*

El reverso Rev(R) de una relación R conteniendo los pares de strings reversos en los pares de R es también regular, pues su traductor aceptor es derivado desde T(R) generalizando el autómata finito estándar para el lenguaje reverso.

Teorema 1.44 *Consideremos un par de lenguajes regulares L_1 y L_2 y supongamos sin restricción de generalidad que sus alfabetos son iguales, entonces la relación $L_1 \times L_2$ es también regular.*

Demostración: Se definen por $M_1 = (\Sigma, Q_1, q_1, F_1, \delta_1)$ y $M_2 = (\Sigma, Q_2, q_2, F_2, \delta_2)$ a los autómatas finitos que aceptan respectivamente los lenguajes L_1 y L_2. Ahora se define el traductor finito.

$$T = (\Sigma, Q_1 \times Q_2, < q_1, q_2 >, F_1 \times F_2, \delta)$$

donde δ esta definida para cualesquiera $s_1 \in Q_1$ $s_2 \in Q_2$ y a y $b \in \Sigma^\lambda$:

$$\delta\left(\langle s_1, s_2\rangle, a, b\right) = \delta_1\left(s_1, a\right) \times \delta_2\left(s_2, b\right)$$

Se puede demostrar por inducción en el número de transiciones que para cualesquiera x e y

$$\delta^*\left(\langle q_1, q_2\rangle, x, y\right) = \delta_1^*\left(q_1, x\right) \times \delta_2^*\left(q_2, y\right)$$

Este resultado vale trivialmente cuando tanto x como y son λ por la definición de δ^* . Si a y b estan en Σ^λ y u y v en Σ^*, luego, usando la definicion de δ^* y la definición de recién para la máquina del producto cartesiano, se tiene:

$$\begin{aligned}
\delta^*\left(\langle q_1, q_2\rangle, ua, ub\right) &= \delta\left(\delta^*\left(\langle q_1, q_2\rangle; u, v\right), a, b\right) \\
&= \delta\left(\delta_1^*\left(q_1, u\right) \times \delta_2^*\left(q_2, v\right), a, b\right) \\
&= \delta_1\left(\delta_1^*\left(q_1, u\right), a\right) \times \delta_2(\delta_2^*\left(\left(q_2, v\right), b\right) \\
&= \delta_1^*\left(q_1, ua\right) \times \delta_2^*\left(q_2, vb\right).
\end{aligned}$$

Luego, $\delta^*\left(\langle q_1, q_2\rangle, x, y\right)$ contiene un estado final si y sólo si tanto $\delta_1^*\left(q_1, x\right)$ como $\delta_2^*\left(q_2, y\right)$ contienen estados finales, por tanto T acepta exactemente los strings de $L_1 \times L_2$ □

Teorema 1.45 *Dados un par de relaciones regulares R_1 y R_2, la composición $R_1 \circ R_2$ de las relaciones también es regular.*

Demostración: Un par de strings $\langle x, y\rangle$ pertenece a la relación $R_1 \circ R_2$ si y sólo si, para algún string intermedio z, $\langle x, z\rangle \in R_1$ y $\langle z, y\rangle \in R_2$.

Ahora, si se define por $T\left(R_1\right) = \left(\Sigma, Q_1, q_1, F_1, \delta_1\right)$ y $T\left(R_2\right) = \left(\Sigma, Q_2, q_2, F_2, \delta_2\right)$, a los traductores de R_1 y R_2, respectivamente, entonces la composición $R_1 \circ R_2$ es aceptada por el traductor finito definido por:

$$\left(\Sigma, Q_1 \times Q_2, < q_1, q_2 >, F_1 \times F_2, \delta\right)$$

donde:

$$\delta\left(\langle s_1, s_2\rangle, a, b\right) = \{\langle t_1, t_2\rangle | \text{ para algun c} \in \Sigma^\lambda, t_1 \in \delta_1\left(s_1, a, c\right) \text{ y } t_2 \in \delta_2\left(s_2, c, b\right)\}$$

En esencia, el δ de la máquina de la composición se forma cancelando los símbolos de cinta intermedios de las transiciones correspondiente en las máquinas componentes. Por inducción en número de transiciones hechas después última cancelada, se sigue que para cualquier par de strings x e y:

$$\delta^*\left(\langle q_1, q_2\rangle, x, y\right) = \{\langle t_1, t_2\rangle | \text{ para algún } z \in \Sigma^*, t_1 \in \delta_1^*\left(q_1, x, z\right) \text{ y } t_2 \in \delta_2^*\left(q_2, z, y\right)\}$$

El traductor de la composición llega al estado final sólo en el caso de que ambas máquinas componentes lo hagan para algún estado intermedio z. Por tanto la composición de dos relaciones regulares es también regular. □

Observación 1.46 *1. La composición de relaciones regulares, como la composición de relaciones en general, es asociativa, esto es* $(R_1 \circ R_2) \circ R_3 = R_1 \circ (R_2 \circ R_3) = R_1 \circ R_2 \circ R_3$

2. $Rango\left(R_1 \circ R_2\right) = Rango\left(R_1\right) / R_2$

Proposición 1.47 *Si L es una relación regular y R es una relación regular cualquiera, entonces los lenguajes R/L y L/R son ambos regulares.*

Demostración Si L es un lenguaje regular entonces existe la relación regular $Id\left(L\right)$ que manda a todos los miembros de L en si mismos. Como $L = Rango\left(Id\left(L\right)\right)$, se tiene:

$$L/R = \left(Rango\left(Id\left(L\right)\right)\right)/R = Rango\left(Id\left(L\right) \circ R\right)$$

Observando que como $Id\left(R\right) \circ R$ es regular y ya se ha visto que el rango de toda relación regular es regular se tiene que L/R es regular. Por simetría del argumento se tiene que también R/L es regular.

□

La clase de relaciones regulares es por definición cerrada bajo las operaciones de unión, concatenación y concatenación repetida. También vale el lema de Pumping.

Teorema 1.48 *El conjunto de relaciones regulares no es cerrado bajo las operaciones de intersección y complemento*

Demostración: Sean:

$$R_1 = \{\langle a^n, b^n c^* \rangle \,|n \geq 0\}$$

$$R_2 = \{\langle a^n, b^* c^n \rangle \,|n \geq 0\}$$

R_1 y R_2 son regulares porque se pueden definir por medio de las expresiones regulares $(a : b)^*(\lambda : c)^*$ y $(\lambda : b)^*(a : c)$, respectivamente. Sin embargo, la intersección $R_1 \cap R_2 = \{\langle a^n, b^n c^n \rangle \,|n \geq 0\}$ tiene como rango al lenguaje libre de contexto $b^n c^n$, lenguaje que no es posible expresar como intersección de regulares (porque sería regular). Teniendo en cuenta que el rango de relaciones regulares es regular se sigue por tanto que la intersección de relaciones relulares no tiene porque ser regular. Que el complemento tampoco, se sigue inmediatamente de la leyes de De Morgan, pues clausura bajo complementación y unión implicaría clausura bajo intersección.

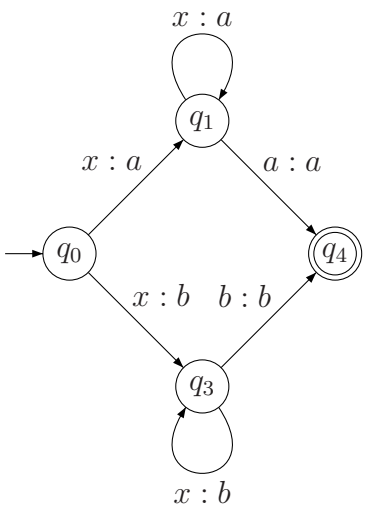
□

Como esto es importante para algunas cuestiones que serán discutidas más adelante, mostramos explícitamente un traductor no determinístico para el que no puede construirse uno determinístico equivalente.

Teorema 1.49 *Existen traductores no determinísticos para los no puede construirse traductores determinísticos equivalentes.*

Demostración 1 Que las relaciones regulares no sean cerradas bajo complementación implica que existen relaciones regulares que sólo son aceptadas por autómatas no determinísticos. Pues si para toda relación regular hay un aceptor determinístico, luego la técnica estándar (teorema 1.22, Hopcroft y Ullman [30]) de intercambiar los estados finales y no finales de este último producirá un traductor finito aceptor de la relación de complementación. □

Demostración 2 Consideremos el traductor de la siguiente figura



Para el input *xxxxxa* debe producir el output *aaaaaa*, mientras que para el input *xxxxxb* el output sera *bbbbbb*. Un traductor finito determinístico equivalente es imposible. Un traductor finito no podría saber cuando devolver a o b mientras vea una x. Sin embargo podrìa no devolver nada y dejar la decisión para más adelante, y, como es una máquina finita, no podrá en general recordar al final cuantas ocurrencias de x ha habido, de manera tal que no sera capaz de imprimir el número correcto de ocurrencias de a y b del inicio.

□

1.2.3. Relaciones regulares con pares strings de la igual longitud

Definición 1.50 *Diremos que una relación regular es de pares de strings de igual longitud (RRSIL) si está compuesta sólo por pares de strings $\langle x, y\rangle$ donde x e y tienen la misma longitud.*

Definición 1.51 *Dado un traductor finito T un string de caminos es una secuencia, eventualmente vacía, de pares de símbolos $(u_1 : v_1)(u_2 : v_2)\ldots(u_n : v_n)$ que marcan las transiciones de un camino de aceptación de T. El Lenguaje de recorridos de T, notado por $Rec(T)$, es el conjunto de todos los strings de recorridos de T*

$Rec(T)$ es regular porque es aceptado por la autómata finito construído directamente de T interpretando las transiciones como elementos de un alfabeto de pares de símbolos no analizables. Análogamente si P es un autómata finito que acepta un lenguaje de pares de símbolos, se define la relación de recorrido $Rel(P)$ como

la relación aceptada por el traductor finito construído de P reinterpretando cada uno de los símbolos de los pares como el par correspondiente de la marca de una transición de un traductor. Es claro de las definiciones que:

$$Rel\left(M\left(Rec\left(T\right)\right)\right) = R\left(T\right)$$

Supongamos ahora que R_1 y R_2 son relaciones regulares aceptadas por los traductores T_1 y T_2, respectivamente. Notemos que $Rec(T_1) \cap Rec(T_2)$ es de hecho un lenguaje regular de pares de símbolos aceptados por algún autómata finito P. Luego $Rel(P)$ existe como una relación regular, inclusive $Rel(P) \subseteq R_1 \cap R_2$. Esto es así porque todo par de strings perteneciente a la relación de recorridos es aceptado por un traductor con un string de camino perteneciendo al lenguaje de recorridos tanto de T_1 como T_2. Luego todo par pertenece tanto a R_1 como a R_2.

La contención opuesta no vale en general. Sin embargo si R_1 y R_2 son aceptados por traductores finitos que no contengan caminos conteniendo transiciones con λ entonces el par de strings peteneciendo a ambas relaciones será aceptados por recorridos marcados idénticamente en ambos traductores, el lenguaje $Rec(T_1) \cap Rec(T_2)$ contendrá el string-recorrido correspondiente a ese par,luego este para pertenece a $Rel(P)$ y, por lo tanto, $Rel(P) = Rec(T_1) \cap Rec(T_2)$.

Lema 1.52 *R es una RRSIL si y sólo si es aceptada por un traductor finito libre de transiciones vacías (TL-λ)*

Demostración Las transiciones de un TL-λ ponen a los símbolos de los strings que acepta en una correspondencia uno a uno , luego, $R\left(T\right)$ es RRSIL.

La prueba en la otra dirección se hace suponiendo que R es una relación regular de la misma longitud aceptado por algún traductor T que tiene transiciones de la forma $u : \lambda$ o $\lambda : v$ (con v y u no iguales a λ pues las transiciones del tipo $\lambda : \lambda$ pueden ser eliminadas por la generalización del algoritmo correspondiente para autómatas finitos). Ahora se eliminan sistemáticamente todas las transiciones conteniendo λ en una secuencia finita de pasos, cada uno de los cuales preserva la relación que es aceptada.

Un recorrido de un estado inicial a otro dado no final contendrá algún número de transiciones de tipo $u : \lambda$ y otro número de transiciones del tipo $\lambda : v$, sin que ambos números sean necesariamente iguales. Observemos que la diferencia entre estos números debe de ser constante para todos los caminos que lleguen al mismo

estado pues la discrepancia debe ser revertida para cualquier camino que partiendo de este estado llegue a uno final. Se puede, por tanto, definir el balance como la diferencia entre número de transiciones de tipo $u : \lambda$ y el número de transiciones del tipo $\lambda : v$ de cualquier recorrido que llegue a ese estado. Observemos también que como un recorrido acíclico no puede producir un balance que difiera de cero en mas de la cantidad de estados del traductor, se tiene que el balance esta acotado por el tamaño del traductor.

En cada iteración el procedimiento tendrá el efecto de remover los recorridos con balance máximo.

Transiciones de la forma $u : v$ conectan estados con el mismo balance. Tales transiciones pueden ser eliminadas en favor de secuencias equivalentes con transiciones de tipo $u : \lambda$ y $\lambda : v$ a un nuevo estado cuyo balance es uno uno menos que el de los estados originales.

Supongase ahora que $k > 0$ es el número máximo de balance para la máquina y que todas las transiciones del tipo $u : v$ entre estados de balance k han sidos removidas. Si q en un estado de balance k este recibirá entradas por transiciones de la forma $u : \lambda$ para estados de balance $k-1$ y tendrá salida hacia estados de balance $k-1$ solamente por transiciones de la forma $\lambda : v$.

Para todas las transiciones $u : \lambda$ de un estado p a q y todas las transiciones $\lambda : v$ de q a r , se construye una nueva transicion $u : v$ de p a r. Luego se remueve el estado q junto con las transiciones que entran o salen de él. Estas manipulaciones no cambian la relación aceptada pero reducen en uno el número de estados con balance k. Se repite este procedimiento para todos los estados con balance k y luego para todos los de balance $k-1$ y así hasta que no queden estados con balance positivo. Un procedimiento simétrico puede ser empleado para estados con balance negativo. Asi se construye un TL-λ, T', equivalente.

□

Observación 1.53 *Es obvio, por las definiciones, que las relaciones regulares son cerradas bajo unión, concatenación, composición, inverso, reverso pues todas esta operaciones preservan tanto regularidad como longitud de strings.*

Teorema 1.54 *Dadas dos RRSIL R_1 y R_2 entonces $R_1 - R_2$ es una relación regular.*

Demostración: Sean T_1 y T_2 los TL-λ aceptores de R_1 y R_2, respectivamente.

Se construye un autómata finito P que acepta el lenguaje de pares de símbolos $Rec\,(T_1) - Rec\,(T_2)$. Un string formado por pares pertenece a la relación $Rel\,(P)$ si y solo si tiene un recorrido de aceptación en T_1 pero no en T_2. Luego $Rel\,(P) = R_1 - R_2$ y por lo tanto es un RRSIL.

□

1.2.4. Algunos operadores importantes

Definición 1.55 *Operador Intro.*

Dado un alfabeto Σ y un conjunto S, se define el operador:

$$Intro\,(S) = [Id\,(\Sigma) \cup [\{\lambda\} \times S]]^*$$

Este operador introduce libremente símbolos en S. Por ejemplo si $a, b \in \Sigma$ y $S = \{@\}$ entonces $Intro(S)$ contiene un comjunto infinito de pares de símbolos, entre ellos $\langle a, a\rangle$,$\langle a, @a\rangle$,$\langle a, a@@@\rangle$ y $\langle a, @@a@b@@\rangle$.

Se nota por $Intro(S)^{-1}$ al operador que remueve todos los elementos de S de un string si S y Σ son disjuntos.

Definición 1.56 *Operador Ignore.*

Dado un lenguaje L y un conjunto de simbolos S, se define el operador $Ignore()$ por:

$$Ignore(S) = L_S = Rango\,(Id(L) \circ Intro\,(S))$$

L_S, el lenguaje L ignorando S, es el lenguaje consistente de los strings procedentes de strings de L en los que los símbolos de L aparecen interspaciados por una cantidad libre de signos de S. Incluye solo strings que deberían de estar en L si algunas ocurrencias de símbolos en S fueran ignoradas.

Definición 1.57 .

Dados dos lenguajes L_1 y L_2 regulares se definen:

$$If\,P\ then S\,(L_1, L_2) = \overline{L_1 \overline{L_2}}$$

$$If\,S\ then P\,(L_1, L_2) = \overline{\overline{L_1} L_2}$$

$$P\ \textit{iff}\ S\,(L_1, L_2) = If\,P\ then S\,(L_1, L_2) \cap If\,S\ then P\,(L_1, L_2)$$

Observemos que por definición como L_1 y L_2 son lenguajes regulares y la definición de los dos operadores de arriba involucra sólo complementación e intersección entonces los lenguajes resultantes son regulares.

De manera mas explícita las definiciones de los dos operadores se pueden escribir en términos de los strings de los lenguajes resultantes:

$If\,P\,then S\,(L_1, L_2) = \{x|$ para toda partición x_1x_2 de x entonces si $x_1 \in L_1, $se tiene que $x_2 \in L_2\}$

Es decir un string es $If\,P\ then S\,(L_1, L_2)$ si todo prefijo en L_1 es seguido por un subfijo en L_2 .

Análogamente:

$If\,S\,then P\,(L_1, L_2) = \{x|$para toda partición x_1x_2 de x entonces si $x_2 \in L_2, $se tiene que $x_1 \in L_1\}$

La doble complementación en la definición de estos operadores condicionales ya en muchos otras expresiones que serán utilizadas más adelante constituyen un medio para expresar cuantificaciones universales. Mientras que una expresión regular $\alpha\beta\gamma$ expresa la proposición de que una instancia de β ocurre entre algunas instancias de α y alguna instancia de γ , la expresión $\overline{\alpha\overline{\beta}\gamma}$ implica que una instancia de β interviene en cada instancia de α y la instancia siguiendo γ.

1.3. Reglas fonológicas generativas en SPE

Las reglas fonológicas son expresiones formales que describen cambios en la representación fonológica de las palabras. Como resultado de la aplicación de una regla, un segmento puede ser borrado o insertado, o uno o más rasgos fonológicos del mismo pueden cambiarse. En este apartado describiremos la notación introducida para estas transformaciones por Chomsky y Halle en 1968 en el libro Sound Pattern of English (SPE).

1.3.1. Representación de los segmentos en SPE

En las representaciones de SPE los segmentos son listas de rasgos. Tal lista refiere a una matriz de rasgos binarios que contine todas las especificaciones de los segmentos que se oponen por alguno de estos rasgos en el sistema fonológico de la lengua. En la propuesta original todos los rasgos eran binarios y no existían dependencias entre los mismos. Por ejemplo una matriz de especificación de una [t] apical es:

$$\begin{bmatrix} -\text{silábico} \\ +\text{consonántico} \\ -\text{sonante} \\ -\text{continua} \\ -\text{distención} \\ +\text{anterior} \\ +\text{coronal} \\ -\text{alto} \\ -\text{bajo} \\ -\text{posterior} \\ -\text{redondeado} \\ +\text{tenso} \\ -\text{sonora} \end{bmatrix}$$

Esta representación se puede abreviar utilizando el rasgo [+*coronal*] cuando se quiera hacer referencia a el o los segmentos especificados por ese valor en el rasgo. Un morfema se representa por un string de matrices de rasgos. El principio y el final del morfema se indican por los símbolos especiales que marcan los límites del morfema. Lo que puede, en retrospectiva, ser complicado pues los únicos límites que reconoce SPE son los límites morfosintácticos. El símbolo + fue incluído como representación

de los límites de morfemas internos de una palabra, mientras que # se usa como límite de palabra.

1.3.2. Reglas fonológicas en SPE

Las reglas fonológicas de reescritura de SPE tienen cuatro partes y su forma general es:

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

Esto significa que el string ϕ será reemplazado por el string ψ cada vez que se encuentre precedido por el string ξ y seguido por el string ρ. En esta notación, si tanto ξ como ρ son vacíos la regla se puede escribir:

$$\phi \rightarrow \psi$$

Los contextos ξ y ρ son generalmente expresiones regulares sobre un alfabeto básico de segmentos. Así se hace posible escribir una regla de armonía vocálica que reemplaza una vocal con el rasgo de posterioridad parametrizado según si la vocal precediendo inmediatemanete la sílaba que la contiene es anterior o posterior. Usando el operador de clausura de Kleene se puede especificar que el número de consonantes que separa a las vocales en cuestión es arbitrario. Si B_i es la contraparte con respecto a la posterioridad de la vocal V_i y B_j es una vocal posterior (tal vez diferente de B_i), esta regla se puede expresar:

$$V_i \rightarrow B_i / B_j C^* \_$$

Existe poco acuerdo acerca de las restricciones que deben aplicarse a ξ y a ρ, nos referiremos a esta porción como al centro de la regla. Son usualmente strings y algunos teóricos las restringen a elementos simples. Estas restricciones no tienen consecuencias matemáticas y se estará abierto a todas las versiones de la teoría mientras que estos elementos denotan lenguajes regulares arbitrarios. Las reglas se pueden especificar también en función de los rasgos del segmento que cambian:

$$\begin{bmatrix}\text{especificación}\\ \text{minimal}\\ \text{del segmento}\end{bmatrix} \rightarrow \begin{bmatrix}\text{rasgo que}\\ \text{cambia}\end{bmatrix} / \begin{bmatrix}\text{especificación}\\ \text{minimal del contexto}\\ \text{a izquierda}\end{bmatrix} \_\_ \begin{bmatrix}\text{especificación}\\ \text{minimal del contexto}\\ \text{a derecha}\end{bmatrix}$$

Se supone que las reglas se aplican, a menos que se especifique, dentro del límite de palabra.

Ejemplo 1.58 *Ensordecimiento de las obstruyentes a final de palabra en el holandés:*

$$[-sonora] \rightarrow [-voz] \,/\_\_\#$$

Esta regla especifica que a final de palabra las obstruyentes se ensordecen. Por ejemplo, la /d/ subyacente se realiza como [t] en:

/pad/ → [pat]: 'sapo'.
/pad/ + ən → [padən]: 'sapos'

Como el objetivo es mostrar que estas expresiones son relaciones regulares, no se permiten reglas del tipo:

$$\lambda \rightarrow ab/a\_b$$

Pues, si se permiten reglas de este tipo, esta regla, donde la salida es el contexto de aplicación, se mapearía el conjunto $\{ab\}$ en el lenguaje libre de contexto $\{a^n b^n, n \geq 1\}$, lo cual está más allá del poder de la relaciones regulares. Sin embargo se permite al material producido por una regla ser el contexto de otra como ocurre, por ejemplo, en el caso de la armonía vocálica.

Se observa, también, que los resultados de aplicar una regla, varían de acuerdo a donde se apliquen sobre el string.

Si se considera la regla:

$$a \rightarrow b/ab\_ba$$

aplicada al string: *abababababa*

Según sea la interpretación se obtendrían tres reglas diferentes:

1) Se aplica la regla a la posición elegibe que este más a la izquierda. En nuestro caso la salida será: *abbbabbbaba*

2) Lo mismo que en 1) pero en sentido contrario: En este caso entonces tendríamos: *ababbbabbba*

3)Se identifican todos los contextos en que se puede aplicar la regla y se aplica en todos simultáneamente. La salida, en el ejemplo es *abbbbbbbbba*.

Se considerarán solamente estas tres estrategias de aplicación de reglas (otras variantes pueden obenerse de estos tres tipos) y se supondrá que cada regla lleva una marca que identifica cual estrategia se usa en su aplicación.

Por otra parte las reglas pueden ser de aplicación obligatoria u opcional. Ambos tipos de reglas producen como mínimo una salida. Observemos sin embargo, que ciertas reglas pueden no producir salidas. Consideremos, por ejemplo, la regla:

$$\lambda \rightarrow b / b\_$$

Aplicando esta regla a un string conteniendo una b, las bs insertadas serán los contextos para sucesivas aplicaciones de la misma regla y así la salida no será un string finito. Este tipo de regla son útiles, sin embargo, para eliminar derivaciones no deseadas.

En contraste con las reglas obligatorias, las opcionales producen típicamente muchas salidas. Por ejemplo, si la regla de arriba ($a \rightarrow b / ab\_ba$) se considera operando de izquierda a derecha y se aplica otra vez al string *abababababa*, algunas de sus salidas podrían ser: *abbbabababa*; *ababbbababa*

Kay y Kaplan [34] citan una regla que debe aplicarse en sentidos diferentes en dos lenguas. En principio para que un traductor que modele una regla de izquierda a derecha modele la regla en sentido inverso pareciera tener que leer en sendido inverso. Sin embargo, se verá que reglas en sentido opuesto no dan traductores también operando en sentido opuesto. En vez de esto la dirección de la regla da cual de las dos cintas (input y output) debe ser chequeada primero. En una regla de izquierda a derecha, el contexto a izquierda de la regla debe ser chequeado contra el input y el contexto a derecha debe ser chequeado contra la porción del string que todavía no ha cambiado pero puede eventualmente ser modificada por aplicaciones de la regla más hacia la derecha del lugar de aplicación actual. En el otro sentido la situación es exactamente la inversa.

El símbolo # es especial en el formalismo de SPE pues sólo puede aparecer en el contexto de una regla. Más adelante se verá que a pesar de requerir un tratamiento especial, también denota relaciones regulares.

Se considerán también reglas actuando de manera conjunta, reglas en Batch, constituídas por un conjunto de reglas que actúan como una única entidad, donde las reglas individuales que la constituyen no tiene preeminencia entre sí. Por ejemplo si consideramos las dos reglas de armonía vocálica en turco deben considerarse una única regla para que modele el fonómeno fonológico referido:

$$\begin{cases} V_i \rightarrow B_i / B_j C^* \_ \\ V_i \rightarrow F_i / F_j C^* \_ \end{cases}$$

Aquí, F representa que una vocal anterior que se opone en el rasgo de posterioridad al conjunto de vocales posteriores.

Esta reglas se pueden también escribir en notación de rasgos:

$$\begin{bmatrix} +silabica \\ -consonante \end{bmatrix} \rightarrow \begin{bmatrix} +posterior \end{bmatrix} \; / \; \begin{bmatrix} +posterior \\ +silabica \\ consonate \end{bmatrix} \begin{bmatrix} +consonante \end{bmatrix}^* \_\_$$

$$\begin{bmatrix} +silabica \\ -consonante \end{bmatrix} \rightarrow \begin{bmatrix} -posterior \end{bmatrix} \; / \; \begin{bmatrix} -posterior \\ +silabica \\ consonate \end{bmatrix} \begin{bmatrix} +consonante \end{bmatrix}^* \_\_$$

En este caso la regla debe aplicarse de izquierda a derecha considerando el string como un todo, aplicándolo en cada posición cuando sea posible.

Es muy común el uso de reglas con variables. Por ejemplo, para expresar fenómenos de asimilación es conveniente usar variables en vez de escribir varias reglas separadas porque así se logran niveles más altos de generalización. Por eso si dos reglas son idénticas excepto por lo que se refiere a los valores del mismo rasgo, entonces las dos reglas pueden ser sustituidas por una sola. Consideremos por ejemplo el siguiente fenómeno del francés referidas a los clusters de dos consonantes (Schane [48]):

Ejemplo 1.59

$$\begin{cases} bt \rightarrow pt \\ gs \rightarrow ks \\ ds \rightarrow ts \\ kb \rightarrow gb \\ tz \rightarrow dz \\ bsh \rightarrow psh \end{cases}$$

Para expresar esta asimilación podemos usar dos reglas:

$$\left[-sonante\right] \rightarrow \left[+sonora\right] \; / \; \underline{\quad} \; \begin{bmatrix} -sonante \\ +sonora \end{bmatrix}$$

$$\left[-sonante\right] \rightarrow \left[-sonora\right] \; / \; \underline{\quad} \; \begin{bmatrix} -sonante \\ -sonora \end{bmatrix}$$

Sin embargo, es conveniente, trabajar a un nivel de generalización mayor utilizando variables:

$$\left[-sonante\right] \rightarrow \left[\alpha sonora\right] \; / \; \underline{\quad} \; \begin{bmatrix} -sonante \\ \alpha sonora \end{bmatrix}$$

1.4. Las Reglas fonológicas como relaciones regulares. La demostración de Kaplan y Kay

El operador Replace se define por:

$$Replace = [Id(\Sigma^*)\ Opt(\varphi \times \psi)]^*$$

El asterisco final permite la repetición del mapeo básico $\varphi \times \psi$ e $Id(\Sigma^*)$ permite que substrings correspondientes idénticos aparezcan entre sucesivas aplicaciones de la regla. El reemplazo $\varphi \times \psi$ es opcional de manera tal de permitir la posibilidad de que la regla en si misma sea opcional o que puedan existir instancias no seleccionables de φ en el string input. La relación *Replace* es el conjunto de pares de strings tal que son idénticas excepto por posibles reemplazos de substrings pertenecientes a φ por substrings perteneciente a ψ. Este relación contiene claramente todos los pares que satisfacen la regla, pero también quizás otros. El problema en lo que sigue consistirá en imponer restriciciones en los mapeos de manera tal que la aplicaciones apliquen en los contextos correctos y de acuerdo con los parámetros fijados por la regla en cuestión. Esto se hará en una serie de aproximaciones.

1.4.1. Requerimientos de contexto

Como primer paso, se pueden agregar simplemente la restricciones de contexto como condiciones necesarias para que se realice el reemplazo $\varphi \times \psi$

$$Replace = [Id(\Sigma^*)\ Opt(Id(\xi)\,\varphi \times \psi Id(\rho))]^*$$

Esta relación incluye strings donde el reemplazo $\varphi \times \psi$ ocurre sólo cuando preceden y suceden inmediatamente substrings satisfaciendo ξ y ρ, respectivamente. Pero esta formulación no toma en cuenta el hecho, observado más arriba, de que los strings de contexto de una aplicación pueden superponerse tanto al contexto o como al centro del string de otro. Por ejemplo si consideramos la siguiente regla opcional que permite que una B abstracta se escriba como b en posicion intervocálica:

$$B \rightarrow b / V\_\_V$$

Con la definición de *Replace* dada arriba el par de strings de 1 de abajo será aceptado pero esto no ocurrirá con el de 2 :

```
1.  V  B V  B V
    V  b V  B V

2.  V  B V  B V
    V  b V  b V
```

Sin embargo el segundo par representa una aplicación válida de la regla porque la vocal central esta sirviendo de contexto a derecha en una aplicación y de contexto a la izquierda en la otra. El problema consiste en que dado un símbolo de un string este puede tener diferentes roles en la aplicacion de una regla, y todas las aplicaciones deben ser tenidas en cuenta.

Como nueva aproximación, se supera esta confusión tomando en cuenta todos estos roles. Primero se considerará como aplicar una regla a strings que han sido preprocesados de manera tal que toda instancia del contexto a izquierda ξ este seguida por un símbolo auxiliar $<$ y toda instancia del contexto a derecha ρ sea precedida por un símbolo auxiliar $>$ donde $<$y$>$ no pertenecen a Σ. Esto hará que el operador de reemplazo puede ser definido solamente en términos de estos marcadores

de contexto, sin importar lo que ξ y ρ realmente especifican y lo que puedan tener en común entre si o con ϕ y ψ. Por ejemplo, para los tres pares de strings que siguen, se asumirá que la relación de reemplazo para los strings superiores de abajo y que los tres strings son aceptados porque en cada uno de los correpondiente pares $B - b$ estan encerrados por <y >:

```
1. >V<B>V<B>V<  2.>V<B>V<B>V<   3.>V<B>V<B>V<
   >V<b>V<b>V<    >V<b>V<B>V<     >V<B>V<b>V<
```

Para tomar un ejemplo más realista consideremos el de realización de las nasales seguidas por labiales:

$$N \rightarrow m/\_\_[+labial]$$

si se aplica al string *iNprobable*, el string preprocesado debe contener la secuencia:

$$< i < N <> p < r < o <> b < a <> b < l < e >$$

El contexto a izquierda de la regla es vacío y por lo tanto hay un marcador de contexto a izquierda < después todo caracter del string original. Toda labial es una instacia del contenido a derecha y de manera acorde hay un > precediendo inmediatamenente las p y b. La regla se aplica propiamente para reescribir la N porque esta se encuentra encerrada por los signos <y >. Por otra parte, si consideramos la versión preprocesada de *iNtratable*,

$$< i < N < t < r < a < t < a <> b < l < e <$$

El > buscado no se encuentra y por tanto la regla no se aplica.

La definición del operador de reemplazo debe ser modificada de dos maneras para operar sobre tales strings preprocesados:

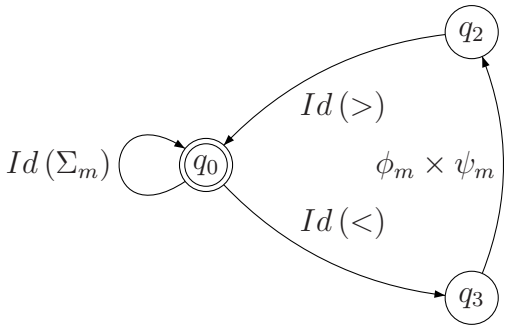
1. Debe permitir que el mapeo $\varphi \times \psi$ sólo ocurra entre los marcadores de contexto adecuados.

2. Algunas ocurrencias de los strings de contexto a derecha y a izquierda no resultarán en la aplicación de la regla o porque la regla es opcional o porque no son satisfechas las otras condiciones de la regla. Luego, la relación debe ignorar

los marcadores correspondientes a esas ocurrencias dentro de los strings identidad entre las aplicaciones de la regla. Relaciones con este comportamiento pueden ser obtenidas por medio del operador *Ignore* que fue definido en la seccion 1.2.3. Este operador se nota con un suscripto que indica el conjunto de símbolos que se ignoran. Si $m = \{<, >\}$, la próxima definición del operador de reemplazo será entonces:

$$Replace = [Id(\Sigma_m^*)\ Opt(Id(<)\,\varphi_m \times \psi_m Id(>))]^*$$

Esto permite que strings arbitrarios de simbolos de $\Sigma \cup \{<, >\}$ sean matchaedos entre aplicaciones de la regla y, además, requiere $<:<$ y $>:>$ para realizar un reemplazo $\varphi \rightarrow \psi$. Los conjuntos m suscriptos también indican que $<$ y $>$ pueden ser ignorados en el medio de un reemplazo, pues la aparición de strings de contexto a izquierda o a derecha es irrelevante en el medio de una aplicación dada de una regla. En la figura a continuación se muestra una representación del traductor del operador de reemplazo. El estado de inicial es q_0 y las transiciones que se muestran son las únicas a partir de las cuales es posible alcanzar el estado final.



Ahora debemos definir relaciones que garanticen que los marcadores de contexto aparezcan efectivamente en los strings donde el operador *Replace* debe aplicarse, es decir sólo cuando esté ratificado por instancias de ξ y ρ. Esto se hará en dos pasos:

Primer Paso

Se usarán relaciones simples para introducir un operador *Prologue*:

$$Prologue = Intro\,(m)$$

Este operador, por definición, introduce libremente los marcadores en m. Un output de *Prologue* es casi lo mismo que el string input pero difiere en el agregado arbitraro de los marcadores. Con la relación $Prologue^{-1}$ se remueven todos los marcadores que aparecen en este output.

Segundo Paso

Se definen relaciones de identidad más complejas que aparean un string con sì mismo si y sólo si estos marcadores aparecen en el contexto apropiado. El operador *P-iff-S* es el componente clave de estos identificadores de contexto. La condición que se debe imponer al contexto por izquierda es que el marcador de contexto por izquierda $<$ aparezca si y sólo si esta inmediatamente precedido por una instacia de ξ . Este requerimiento básico es satisfecho por los strings en el lenguaje regular:

$$P\text{-}iff\text{-}S\,(\Sigma^*\xi, < \Sigma^*)$$

La solución es, sin embargo, un poco más complicada por dos circunstancias que se deben tener en cuenta:

1. Una instacia de ξ puede tener prefijos que sean también instancias ξ. Si ξ es una expresión como ab^* entonces $a < b <$ es una marca aceptable pero $ab <$ y $a < b$ no lo son porque las dos instancias de ξ no están seguidas por $<$. Los marcadores que necesariamente siguen tales prefijos no deben impedir que instancias más grandes sean también identificadas y marcadas. Por otra parte, tampoco los marcadores de contexto a derecha deben interferir en la marcación de contextos a izquierda. Para lograr esto, se utiliza el operador $Ignore$ y se define:

$$Piff\S\,(\Sigma^*_<\xi_<, < \Sigma^*_<)_>$$

   Así, sin embargo, se pasan por alto muchos marcadores: como una instancia de $\xi_<$ seguida por $<$ es también una instancia de $\xi_<$ entonces también debe de

estar seguida por un marcador y así sucesivamente. Así, los únicos strings finitos que pertenecen a este lenguaje serían aquellos que no contiene finalmente ninguna instancia de ξ. Para identificar y marcar correctamente contextos a izquierda los marcadores que sigan a una instancia de $\xi_<$ no deben ser ignorados. Luego, agregando este último requisito se define el lenguaje regular $ContextIzq\,(\xi, <, >)$ donde el operador de contexto a izquierda se define por:

$$ContextIzq\,(\xi, <, >) = PiffS\,(\Sigma_l^*\xi_l - \Sigma_l^* l, l\Sigma_l^*)_>$$

Conviene parametrizar esta operación con el formato del contenido a izquierda y con los marcadores en uso de manera tal que pueda ser usada más adelante.

2. La segunda complicación proviene de las reglas que son para insertar o borrar material en el string, en estos tanto ϕ como ξ pueden incluir al string vacío λ. Consideremos la regla de izquierda a derecha:

$$a \rightarrow \lambda / b\_\_$$

La aplicación iterada de esta regla puede borrar una secuencia arbitraria de caracteres a, convirtiendo por ejemplo strings de la forma $baaaaaaaaa...a$ en el stritg b. La b en el comienzo del string inicial sirve como contexto de la regla en cada aplicación de la misma. Esto presenta un problema para la aplicación que hemos construído más arriba pues la relación *Replace* requiere un marcador $<$ distinto para cada aplicación de la regla y el $<$ que sanciona la deleción de la a más hacia la izquierda no estará dìsponible para borrar la próxima a. Sin embargo, el operador *ContextIzq* según ha sido definido no permite dos marcadores de contexto a izquierda continuados. La solución consiste en insertar un caracter explícito 0 para representar el material borrado. Si *ContextIzq* ignora este caracter en ξ, 0 puede ser siempre seguido por otro marcador a izquierda y así se permite otra aplicación de la regla.

El símbolo auxiliar 0 no está en Σ o en el conjunto de marcadores. Sustituirá los strings vacíos que pueden aparecer en el centro de las reglas ϕ o ξ, pero es un símbolo genuino en un alfabeto expandido a diferencia del string vacío que que aparece como un caracter distintivo como un elemento distintivo en los strings de caracteres. La relación *Prologue* se extiende para que introduzca libremente 0 y marcadores de m

$$Prologue = Intro\,(m \cup \{0\})$$

Se construyen ahora versiones alternativas de ϕ y ψ para las cuales este símbolo especial reemplaza los verdaderos strings vacios:

$$\phi^0 = \begin{cases} \phi & \text{si } \lambda \notin \phi \\ [\phi - \lambda] \cup 0 & \text{en otro caso} \end{cases}$$

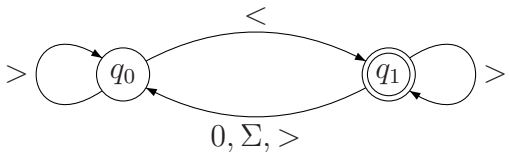
esta contiene exactamente los mismos strings que ϕ salvo que el singleton 0 como string es incluído en lugar de el string vacío que de otra manera tendría que estar en el lenguaje. ψ^0 se define análogamente y por tanto el operador *Replace* se expesara en términos de estos lenguajes regulares por:

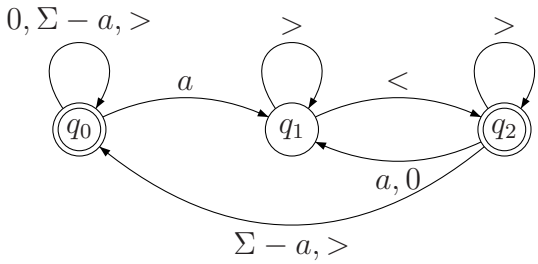
$$Replace = \left[Id\,(\Sigma^*_{m\ 0})\ Opt(Id\,(<)\,\phi^0_m \times \psi^0_m Id\,(>))\right]^*$$

Se completa la definición del identificador de contextos a la izquierda:

$$ContextIzq\,(\xi, l, r) = P\text{-}sii\text{-}S(\Sigma^*_{l\ 0}\xi_{l0} - \Sigma^*_l\, l, l\Sigma^*_{l0}))_r$$

Tal cual nuestro objetivo el lenguaje notado por este operador incluye strings si y sólo si todo substring perteneciente a ξ (ignorando l, r y 0) esta seguido por un marcador con l. Este efecto es ilustrado por los diagramas de los siguientes traductores:





La primer máquina es el aceptor minimal (en cantidad de estados) para el lenguaje de contexto vacío $ContextIzq(\lambda, <, >)$. Acepta strings que tienen al menos un $<$ y todo 0 o símbolo de Σ que debe ser seguido inmediatamente por un $<$. El marcador $>$ de transiciones representa el hecho de que $>$ es ignorado. La segunda máquina acepta el lenguaje $ContextIzq\,(a, <, >)$, y requiere que los $<$ aparezcan después de toda a o depués de todo 0 que siga a una a. Esta máquina en particular es no determinística para que sea más fácil entender su funcionamiento.

Un operador para identificar y marcar contextos a izquierda puede ser definido simétricamente:

$$ContextDer\,(\rho, l, r) = P\text{-}iff\text{-}S(\Sigma^*_{r\ 0}\mathrm{r}, \rho_{r0}\Sigma^*_{r\ 0} - r\Sigma^*_{r\ 0}))_l$$

Luego $ContextDer\,(\rho, <, >)$ incluye strings si y sólo si todo substring perteneciente a ρ esta precedido inmediatamente por un marcador de contexto a derecha $>$. Alternativamente, usando que el hecho de que el reverso de un lenguaje regular es un lenguaje regular, se puede definifr el $ContextDer$ usando el $ContextIzq$:

$$ContextDer(\rho, l, r) = Rev(ContexIzq(Rev(\rho), l, r))$$

Estos identificadores de contexto notan conjuntos apropiados de strings inclusive para reglas con contextos no especificados pues los contextos vacíos son interpretados como si el string vacío fuera el especificado. El string vacío indica que símbolos adyacentes no tienen influencia en la aplicación de la regla. Si, por ejemplo, una ξ omitida es interpretada como λ, todo string de $ContextIzq$ tiene uno y sólo un marcador de contexto a izquierda en su inicio, su final y entre todos los pares de símbolos en Σ, permitiendo la aplicación de la regla en cada posición.

1.4.2. Aplicación simultánea y direccional de las reglas

Una vez que se definieron los componentes que introducen y remueven marcadores de contexto para rechazar strings con marcadores mal ubicados y para representar la acción de una regla entre marcadores apropiados, la relación regular que modela la aplicación opcional de una regla se modela utilizando estas piezas. El orden de composición depende de si la regla es específicamente aplicada iterativamente de izquierda a derecha o viceversa. Como fue notado en la sección 1.3.2 la diferencia es

que para las reglas de izquierda a derecha, el contexto a izquierda, ξ puede matchear contra el output de la aplicación previa de la misma regla, pero el contexto a derecha, ρ, debe matchear contra el string del input sin cambios. Esta situación es modelada directamente por el orden en el que componentes de la regla son combinados. Para una regla de izquierda a derecha, el contexto a la derecha es chequeado en el lado de los reemplazos del input, mientras que el otro contexto es chequeado en el lado de la salida. La relación regular y traductor correspondiente para una regla opcional de izquierda a derecha se modela por la siguiente secuencia de composiciones:

$$
\begin{gathered}
Prologue \circ \\
Id(ContextDer(\rho, <, >)) \circ \\
Replace \circ \\
Id(ContextIzq(\xi, <, >)) \circ \\
Prologue^{-1}
\end{gathered}
$$

Obsérvese que tanto los marcadores de contextos a izquierda como a derecha son introducidos libremente en los strings inputs, los strings en los cuales los marcadores de contexto a derecha son rechazados, y los reemplazos tienen lugar solamente entre los marcadores de contexto a derecha, ya sujetos al contexto, y los todavia libres, MCI. Esto impone sobre los MCI la restricción de aparecer antes que los reemplazos, sin embargo pueden o no aparecer libremente en cualquier lado. El tester de MCI asegura que los mismos aparezcan sólo en las ubicación apropiada en el output. Finalmente, todos los marcadores son eliminados, dando strings en el lenguaje de salida.

La situación de chequeo del contexto se invierte para reglas de derecha a izquierda: los MCI matchean contra el string input sin cambiar mientras que los MCD los hacen contra el output. La aplicación de derecha a izquierda de reglas opcionales puede ser modelada invirtiendo simplemente el orden en que se aplican las relaciones que chequean contextos:

$$Prologue \circ$$

$$Id(ContextIzq(\lambda, <, >)) \circ$$

$$Replace \circ$$

$$Id(ContextDer(\rho, <, >)) \circ$$

$$Prologue^{-1}$$

Observemos que también el traductor correspondiente modelando esta relación regular, modela una regla de derecha a izquierda, mientras se mueve en sentido opuesto sobre sus cintas.

Aplicación simultánea de reglas

La aplicación simultánea de reglas, en las cuales todos los contextos de aplicación son chequeados antes de que la reescritura tenga lugar, se modela por una cascada que identifica tanto los contextos a izquierda como a derecha en el lado del input del reemplazo:

$$Prologue \circ$$

$$Id(ContextIzq(\lambda, <, >) \cap ContextDer(\rho, <, >)) \circ$$

$$Replace \circ$$

$$Prologue^{-1}$$

1.4.3. Reglas de aplicación obligatoria

En este apartado se construirán composiciones que modelen la aplicación obligatoria del reemplazo $\phi \times \psi$ bajo el contexto especificado. Para eso requerimos una condición adicional que rechace strings conteniendo sitios donde la condiciones de aplicación estén dadas pero donde no se haya realizado el reemplazo. Esto es, se debe restringir la relación de manera tal que sin importar por el momento del efecto de aplicaciones que se solapen, todo substring de la forma $\xi\phi\rho$ debe corresponder a un substring de la forma $\xi\psi\rho$ en el segundo par. Se puede definir esta resticción encuadrándola en términos de nuestros marcadores de contexto: el operador *Replace*

no debe contener un par con el substring $< \phi >$ en un elemento en correspondencia con un elemento distinto de $< \psi >$ en el otro.

Un primer intento para formular este requerimiento es el que se obtiene tomando el complemento de la relación que incluye las correspondencias no deseadas, lo que sugiere la relación:

$$\overline{Id(\Sigma^*_{m0})\, Id(<)\, \phi^0_m \times \overline{\psi^0_m} Id(>)\, Id(\Sigma^0_{m0})}$$

Esta expresión puede ser tomada como el punto de partida de varios argumentos que tomarán en cuenta correctamente varias aplicaciones que se solapen. Sin embargo, esta línea no nos permite establecer el hecho de que las reglas obligatorias también definen mapeos regulares. En primer lugar la expresión involucra el complemento de una relación regular y hemos observado que el complemento de una relación regular no es necesariamente una relación regular. En segundo lugar, inclusive si la relación resultante es regular, la manera obvia de poner esta regla en composición sería intersecarla con la relación de reemplazo, y como ya se ha visto, esto no es necesariamente regular.

Por tanto, probar que reglas obligatorias definen mapas regulares requerirá un análisis más cuidadoso de los roles que los marcadores de contexto (MC) pueden jugar en los strings intermediarios involucrados en la composición de las reglas.

Un MCI puede puede servir al operador *Replace* de tres maneras:

1. Puede empezar la aplicación de la regla siempre y cuando aparezca frente de una configuración apropiada de ϕ ψ y MC a derecha (MCD).

2. Puede ser ignorado durante la aplicación idéntica sobre el string, las regiones entre las aplicaciones la relación de reemplazo.

3. Puede ser ignorado porque viene en el medio o centro de una aplicación de otra regla que empezó a la derecha del marcador en cuestión y se extiende más a la derecha.

Supongamos que cada una de las situaciones enunciadas arriba se notan con tres simbolos distintos: $<_a$,$<_i$,$<_c$, respectivamente y también que se definen tres marcadores análogos para MCD.

Dondequiera que una regla sea aplicada propiamente, el lado del input del la relación de reemplazo debe contener un substring de la forma:

$$<_a \ \phi^0_{<_c>_c} \ >_a$$

La diferencia crucial en el caso donde una regla de izquierda a derecha falla en su aplicación es que el contexto a izquierda precediendo a ϕ^0 este marcado con un $<_i$ en vez de una $<_a$ porque es parte de una secuencia identidad. Esta situación es indeseable sin importar que tipos de marcadores sean ignorados en la forma de ψ^0 o marcados por el contexto a derecha de esta potencial aplicación. Si estos marcadores se encuentran al centro o en el borde de los reemplazos que se están llevando a cabo más hacia la derecha de la situación que nos complica, la aplicación más hacia la izquierda, marcada con $<_i$ debe tener precedencia.

Los símbolos $<$ y $>$ fueron anteriormente usados como caracteres auxiliares apareciendo en strings intermedios. Con un leve abuso de notación, haremos que ellos actúen como símbolos permaneciendo en los conjuntos a MC $\{<_a, <_i, <_c\}$ $\{>_a, >_i, >_c\}$, respectivamente. Notaremos por m a la unión de los dos conjuntos precedentes ($< \cup >$). Un substring del lado del input del reemplazo es luego una aplicación pedida de izquierda a derecha si matchea con la fórmula:

$$<_i \ \phi^o_m >$$

Se puede entonces forzar la aplicación obligatoria de una regla de izquierda a derecha requiriendo que los strings del lado del input no contengan tales substrings. O, lo que es lo mismo, que pertenezcan al lenguaje regular $Obligado(\phi, <_i, >)$, donde el operador *Obligado* se define por:

$$Obligado\,(\phi, l, r) = \overline{\Sigma^*_{m0} \ l \ \phi^0_m \ r \ \Sigma^0_{m0}}$$

Por simetria, la aplicación necesitada de la RDI matchea el pattern

$$< \phi^o_m >_i$$

y $Obligado(\phi, <, >_i)$ es el filtro del input apropiado para no permitir esos substrings. Notar que el operador *Obligado* involucra solamente lenguajes regulares y no relaciones de manera tal que resulta regular a pesar de usar complementos.

Se debe ahora arreglar los diferentes tipos de marcadores para que aparezcan en el input de *Replace* sólo en las circunstancias apropiadas. Como antes, los marcadores de contexto deben asegurar que ninguno de los marcadores de contexto aparezcan a

menos que estén precedidos y seguidos por el contexto apropiado y que toda ocurrencia de un contexto este marcada por un marcador libremente elegido del conjunto correspondiente de tres. Los operadores *ConstextDer* y *ConstextIzq* producirán exactamente el efecto deseado cuando sean aplicados sobre el último sentido dado a $<$ y $>$ y m. El operador *Replace* debe ser nuevamente modificado, sin embargo, para que sólo distinga los diferentes roles de los marcadores. La siguiene definición elige los marcadores correctos para todos los parámetros de la aplicación de la regla:

$$Replace = \left[Id\left(\Sigma^*_{<_i>_i 0}\right)\ Opt(Id(<_a)\,\phi^0_{<_c>_c} \times \psi^0_{<_c>_c} Id(>_a))\right]^*$$

La acción de las reglas obligatorias será modelada insertando el filtro apropiado en la secuencia de composiciones. Las reglas de izquierda a derecha son modeladas por la cascada:

$$Prologue \circ$$
$$Id(Obligado(\phi, <_i, >))\ \circ$$
$$Id(ContextDer(\rho, <, >))\ \circ$$
$$Replace \circ$$
$$Id(ContextIzq(\lambda, <, >))\ \circ$$
$$Prologue^{-1}$$

Para reglas de derecha a izquierda por:

$$Prologue \circ$$
$$Id(Obligado(\phi, <, >_i))\ \circ$$
$$Id(ContextIzq(\lambda, <, >))\ \circ$$
$$Replace \circ$$
$$Id(ContextDer(\rho, <, >))\ \circ$$
$$Prologue^{-1}$$

Se debe remarcar inclusive que aunque las reglas obligatorias no producen necesariamente un singleton string como output. Si un lenguaje ψ contiene mas de un string , luego los outputs serán producidos por cada uno de estos strings en cada sitio

de aplicación. Inclusive, si ϕ contiene strings que son sufijos o prefijos (dependiendo de la dirección de aplicación)de otros strings en ϕ, las alternativas serán producidas para cada longitud del matcheo. Un formalismo particular puede especificar como resolver tales ambigüedadades y estas estipulaciones deben ser modeladas por restricciones adicionales en nuestro formalismo. Por ejemplo el requerimiento de que sólo los matcheos ϕ mas cortos sean reescritos puede ser impuesto ignorando sólo uno de los $<_c$ o $>_c$ en la aplicación de *Replace*, dependiendo de la dirección de la misma. Hay diferentes formulaciones para la aplicación obligatoria de reglas simultáneas, también dependiendo de como se resuelva la competencia entre aplicaciones sobre sitios que se solapan. Intersecando los dos filtros obligatorios como sigue modela el caso donde el substring más largo ϕ sea preferido sobre matcheos más cortos que lo solapen:

$$\begin{gathered} \mathit{Prologue} \circ \\ \mathit{Id}(\mathit{Obligado}(\phi, <_i, >) \cap \mathit{Obligado}(\phi, <, >_i)) \circ \\ \mathit{Id}(\mathit{ContextDer}(\rho, <, >) \cap \mathit{ContextIzq}(\rho, <, >)) \circ \\ \mathit{Replace} \circ \\ \mathit{Prologue}^{-1} \end{gathered}$$

Así, los operadores pueden ser redefinidos y combinados de diferentes maneras para modelar otros regímenes de resolución de solapamientos.

1.4.4. Límites de morfemas

Una regla conteniendo al marcador especial # indica que su aplicación esta condicionada por el comienzo o final del string. Un marcador de límite sólo tiene sentido cuando ocurre en las partes del contexto de palabra, especificamente, cuando ocurre en el límite de la izquierda de un string de contexto por la izquierda o bien en el limite derecho de un string de contexto por el otro lado. No es necesario ningún tratamiento especial para el marcador de límite de string si el # aparece como el primer o último caracter apareado de todo string input y output. Si este fuera el caso, la cascadas composicionales de los apartados anteriores modelarían exactamente la interpretación deseada donde la aplicación de la regla sea sensitiva a los

bordes. Strings ordinarios input y output no tienen esta característica, pero una simple modificación de la relación *Prologue* puede simular esta situación. Se extiende la definición del operador por:

$$Prologue = Intro\,(m \cup \{0\}) \circ \left[\lambda : \#\ Id\left(\overline{\Sigma^0_{m0} \# \Sigma^0_{m0}}\right)\ \ \lambda : \#\right]$$

Se ha impuesto una relación adicional que introduce el marcador del límite de string al principio y al final de un string en principio marcado libremente y también se han rechazado los strings que contienen los marcadores de límite de morfema en algún lugar del medio del mismo. El efecto en red es que los strings por debajo de *Prologue* en la cascada tienen los límites marcados, imagenes marcadas de los strings inputs originales y los MC pueden propiamente detectar los bordes de estos strings. El inverso de *Prologue* en el fondo de la cascada de composiciones remueve los marcadores de límite junto con los otros símbolos auxiliares.

1.4.5. Reglas en Batch

En esta sección se modelará la aplicación de reglas que actúan como un conjunto de reglas sin que haya preeminencias, denominadas reglas en Batch. Se recalca que para cada posición en el string input cada regla en el batch es tenida en cuenta independientemente para su aplicación. Existe un tratamiento directo para aproximar este comportamiento.

Para esto, consideremos: $\{R_1, \ldots, R_n\}$ el conjunto de relaciones regulares que deben ser aplicadas como un batch y construyamos la relación:

$$[\cup_k R_k]^*$$

Como esta relación se contruye con la clausura bajo la unión, es regular e incluye todos los pares de strings que son idénticos salvo por substrings que difieren de acuerdo con la rescritura especificada por al menos alguna de las reglas. En particular, esto no permite solapamiento entre material que satisface los requerimientos para la aplicación de una regla del conjunto con los elementos que autorizan la aplicación de otra regla. Como siempre tomamos en cuenta este nuevo arreglo de dependencias que se superponen introduciendo un conjunto de numerosos símbolos especiales que manejaremos cuidadosamente, considerando sus ocurrencias e iteracciones.

Una regla batch es un conjunto de subreglas:

$$\left\{\phi^1 \rightarrow \psi^1 / \xi^1 \_\_ \rho^1, \ldots, \phi^n \rightarrow \psi^n / \xi^n \_\_ \rho^n\right\}$$

junto con una especificación de parámetros estándar de aplicación (de izquierda a derecha, obligatoria, etc.). Para evitar, tanto como sea posible, confusiones con otras notaciones se usarán superíndices para distinguir los componentes de las diferentes reglas. Una parte crucial del tratamiento de una regla ordinaria es introducir un marcador especial para la aparición de los contextos a derecha y a izquierda de manera tal que los reemplazos dados por esta sean llevados a cabo sólo en los contextos apropiados. Aqui se hará los mismo para cada una de las reglas de batch pero usando un conjunto de marcadores diferentes para cada regla. Estos marcadores permiten codificar en un solo string la ocurrencia de sucesos para todas las diferentes subreglas que cada MC de subregla marca distintivamente.

Sea $<^k$ el conjunto $\left\{<_i^k, <_a^k, <_a^k\right\}$ de MCI para la regla $\phi^k \rightarrow \psi^k / \lambda^k \_ \rho^k$, subregla k-ésima del batch. Análogamente $>^k$ es el conjunto de MCD para la misma regla, notemos por m^k, $<$, $>$ y m a:

$$m^k = <^k \cup >^k$$

$$<= \cup_k <^k$$

$$>= \cup_k >^k$$

$$m = < \cup >$$

Observemos que con esta redefinición de m la relación *Prologue* introducirá ahora los marcadores de todas las subreglas. Será de utilidad notar también el conjunto de marcadores que no contienen los de la regla k-ésima:

$$m^{-k} = m - m^k$$

Consideremos ahora el lenguaje regular:

$$ContextIzq\left(\xi^k, <^k, >^k\right)_{m^{-k}}$$

Este lenguaje contiene strings en los cuales todas las instancias de la expresión de CI de la subregla k-ésima es seguida por uno de los k-ésimos marcadores, y tales marcadores aparecen sólo depués de instancias de ξ^k. Los marcadores de contextos k-ésimos son libremente distribuidos, como asi también todos los marcadores para las otras subreglas. La ocurrencias de todos los otros MCI son restingidos en lenguajes regulares definidos similarmente. Poniendo todas estas restricciones de los marcadores en conjunto, obtenemos:

$$\cap_k \mathit{ContextIzq}\left(\lambda^k, <^k, >^k\right)_{m^{-k}}$$

Este lenguaje tiene cada uno las MCI monótonamente marcado por uno de los MCI. Esto deja todos los MCD sin restricciones, serán restringidos a sus propias posiciones por el correspondiente lenguaje para el contexto a la derecha:

$$\cap_k \mathit{ContextDer}\left(\rho^k, <^k, >^k\right)_{m^{-k}}$$

Estos lenguajes obtenidos de intersecciones, ambos regulares, tomarán el lugar de MC simples cuando formemos la cascadas de composiciones para modelar la aplicación de reglas en Batch. Estos MC generalizados son también apropiados para reglas ordinarias si consideramos cada uno de ellos como un batch conteniendo una sola regla.

Se debe construir también un operador de reemplazo para reglas en Batch. Este debe mapear strings en el input a otros en el output con MC ubicados correctamente, asegurando que cualquiera de las subreglas de reescritura se pueda aplicar en cada posición correctamente marcada, pero, además, que la subregla k-ésima ocurra sólo entre $<_a^k$ y $>_a^k$. El total de posibles reescrituras se codifica por medio de:

$$\cup_k \left[\mathit{Id}\left(<_a^k\right) \phi^{0k}_{<_c>_c} \times \psi^{0k}_{<_c>_c} \mathit{Id}\left(>_a^k\right))\right]$$

donde $<_c = <_c^1, \ldots, <_c^k$ el conjunto de todos los marcadores centrales a izquierda. Se incorpora esta relación como parte de una nueva definición del operador *Replace*, con los $<_i$ y $>_i$ representando ahora los conjuntos de todos los marcadores a izquierda y a derecha:

$$\mathit{Replace} = \left[\mathit{Id}\left(\Sigma^*_{<_i>_i 0}\right)\ \mathit{Opt}(\cup_k \left[\mathit{Id}\left(<_a^k\right) \phi^{0k}_{<_c>_c} \times \psi^{0k}_{<_c>_c} \mathit{Id}\left(>_a^k\right))\right])\right]^*$$

Esta relación permite realizar todo reemplazo apropiado separado por subtrings idénticos. Es regular por la propiedad de clausura por unión, lo cual no se podría asegurar si se hubiera usado intersección o complementación en la construcción.

Un modelo para la aplicación de izquierda a derecha, opcional, de un regla de batch se obtiene sustituyendo las nuevas definiciones en la cascada de composiciones para reglas ordinarias :

$$
\begin{gathered}
\mathit{Prologue} \circ \\
\mathit{Id}\left(\cap_k \mathit{ContextDer}\left(\rho^k, <^k, >^k\right)\right)_m^{-k}) \circ \\
\mathit{Replace} \circ \\
\mathit{Id}\left(\cap_k \mathit{ContextIzq}\left(\xi^k, <^k, >^k\right)\right)_m^{-k}) \circ \\
\mathit{Prologue}^{-1}
\end{gathered}
$$

Las reglas en Batch opcionales de derecha a izquierda se pueden modelar de manera análoga substituyendo en las correspondientes cascadas de composiciones para reglas ordinarias para ese sentido. La aplicaciones obligatorias son manejadas combinando instancias del operador *Obligado* construído independientemente para cada subregla: $\mathit{Obligado}\left(\phi^k, <^k, >^k\right)$ excluye todos los strings en los cuales la k-ésima regla no es aplicada y se aplica de izquierda a derecha siempre que sus condiciones de aplicación sean satisfechas. La intersección de los filtros obligatorios para todas las subreglas en el batch asegura que al menos una subregla sea aplicada en cada posición donde pueda aplicarse. Luego el comportamiento de una regla en Batch obligatoria de izquierda a derecha sera modelada por la siguiente cascada:

$$
\begin{gathered}
\mathit{Prologue} \circ \\
\mathit{Id}\left(\cap_k \mathit{Obligado}\left(\phi^k, <^k, >^k\right)\right) \circ \\
\mathit{Id}\left(\cap_k \mathit{ContextDer}\left(\rho^k, <^k, >^k\right)_m^{-k}\right) \circ \\
\mathit{Replace} \circ \\
\mathit{Id}\left(\cap_k \mathit{ContextIzq}\left(\xi^k, <^k, >^k\right)_m^{-k}\right) \circ \\
\mathit{Prologue}^{-1}
\end{gathered}
$$

Nuevamente las sustituciones hechas sobre las cascadas para las reglas correspondientes modelan el comportamiento de la aplicación de derecha a izquierda y simultánea.

1.4.6. Matrices de rasgos y Variables rasgos finitos

Se trabajará ahora con matrices de rasgos con variables que toman valores finitos para ver que son relaciones regulares. Para ejemplificar el modo en que esto se hace consideramos una regla en particular, la regla de armonía vocálica del turco que fue considerada en la sección 1.3.2:

$$\begin{bmatrix} +silabica \\ -consonante \end{bmatrix} \rightarrow \begin{bmatrix} \alpha\ posterior \end{bmatrix} \ / \ \begin{bmatrix} \alpha\ posterior \\ +silabica \\ consonate \end{bmatrix} \begin{bmatrix} +consonante \end{bmatrix}^{*} \_\_\_$$

Reescribiremos esta regla en una notación que si bien es mas engorrosa es más manejable matemáticamente:

$$\begin{bmatrix} +silabica \\ -consonante \\ \beta\ posterior \\ \beta_1 redonda \\ \beta_2 alta \\ \dots \\ \beta_n f_n \end{bmatrix} \rightarrow \begin{bmatrix} +silabica \\ -consonante \\ \alpha\ posterior \\ \beta_1 redonda \\ \beta_2 alta \\ \dots \\ \beta_n f_n \end{bmatrix} \ / \ \begin{bmatrix} \alpha\ posterior \\ +silabica \\ -consonante \end{bmatrix} \begin{bmatrix} +consonante \end{bmatrix}^{*} \_\_\_$$

Las matrices de rasgos input y output están ahora totalmente especificadas y en el contexto el valor de todo rasgo no mencionado puede ser libremente elegido.

Una matriz de rasgos en una expresión regular es muy fácil de interpretar cuando no contiene ninguna variable de rasgos. Tal matriz simplemente abrevia la unión de todos los símbolos segmentales que comparten los rasgos especificados y, por tanto, la matriz puede ser reemplazada por un conjunto de símbolos no analizables sin cambiar el significado de la regla. Luego la matriz: [+*consonante*] puede ser trancripta en el lenguaje regular $\{p, t, k, b, d, \dots\}$ y tratada con técnicas estándar. Eventualmente, si los rasgos son incompatibles la matriz de rasgos será reemplazada por un conjunto vacío de segmentos.

Una simple traslación es también posible para variables de rasgos para las cuales todas sus ocurrencias esten localizadas en sólo una parte de la regla, como en el siguiente contexto ficticio:

$$[\alpha\ alta]\,[+consonate]^*\,[-\alpha\ redonda]$$

Si α toma el valor + entonces la primer matriz en instanciada a $[+alta]$ y denota el conjunto de simbolos no analizables, digamos $\{e, i, \ldots\}$ que satisfacen esa descripción. La última matiz se reduce a $[-redonda]$ y denota otro conjunto de símbolos no analizables (por ejemplo $\{a, e, i, \ldots\}$). Por tanto la expresión entera es equivalente a:

$$\{e, i, \ldots\}\,\{p, t, k, b, d\ldots\}^*\,\{a, e, i, \ldots\}$$

Por otra parte, si α toma el valor $-$ entonces la primera matriz es instanciada a $[-alta]$ denotando un conjunto diferente de símbolos, digamos $\{a, o, \ldots\}$ y la última matriz se reduce a $[+redonda]$. La expresión entera de esta instanciación de α es equivalente a:

$$\{a, o, \ldots\}\,\{p, t, k, b, d\ldots\}^*\,\{o, u, \ldots\}$$

En la interpretación convencional, la expresión original matchea strings que pertenecen a alguno de este lenguajes regulares instanciados. En efecto, la variable es usada para codificar un correlación entre elecciones de difrentes conjuntos de símbolos no analizables. Se puede formalizar esta interpretación de la siguiente manera. Supongamos que θ es una expresión regular sobre las matrices de rasgos conteniendo una variable simple α para un rasgo cuyos valores finitos provienen de un conjunto V , comunmente el conjunto $\{+, -\}$. Sean $\theta\,[\alpha \to v]$ los valores resultantes de sustituir $v \in V$ por α donde esta ocurra en θ y reemplazar luego cada matriz de rasgos de variables libres en este resultado por el conjunto de símbolos no analizables que satisfacen esa descripción de rasgos. La interpretación de θ esta dada por la fórmula

$$\cup_{v \in V} \theta\,[\alpha \to v]$$

Esta traducción produce una expresión regular que modela propiamente la correlación entre las elecciones definidas por α en la expresión original.

Expresiones de reglas conteniendo muchas variables que ocurren localmente pueden ser manejadas por generalizaciones obvias del esquema anterior de sustitucion. Si $\alpha_1, \ldots, \alpha_n$ son las variables locales en θ cuyos valores viene de finitos conjuntos $V_1, \ldots, V_n$, el conjunto de n-tuplas:

$$I = \{\langle \alpha_1 \rightarrow v_1, \ldots, \alpha_n \rightarrow v_n | v_1 \in V_1, \ldots, v_n \in V_n \rangle\}$$

representa la colección de toda las posibles instanciaciones de valores de estas variables. Si dejamos que $\theta\,[i]$ sea el resultado de llevar a cabo las substituciones indicadas por todas las variables para algún $i \in I$, la interpretación de toda la expresión esta dada por la fórmula:

$$\cup_{i \in I} \theta\,[i]$$

Cuando todas la variables locales son trasladadas, la expresión resultante puede todav contener matrices de rasgos con variables no locales, es decir, que pueden ocurrrir en otras partes de la regla. Las expresiones del input y del output tendrán casi siempre variables en común por la expansión del centro introducida en el paso incial. Las variables que aparecen en más de una parte de una regla claramente no pueden ser eliminadas de cada parte independientemente, porque si no fuera así la correlación entre esas instanciaciones se perdería. Una regla de matrices de rasgos debe ser interpretada como escaneando en la dirección apropiada a lo largo del string input hasta que es encontrada una configuración de los símbolos tal que satisface las condiciones de aplicación de las reglas instanciada por una selección de valores para todas sus variables. Los segmentos que matchean el input son luego reemplazados por los segmentos output determinados por la misma selección, y el escaneo continua hasta que otra configuración apropiada es localizada (con, eventualmente, una selección de diferentes valores para las variables). Este comportamiento es modelado por una aplicación en el modo de batch de un conjunto de reglas, cada una de las cuales corresponde a una instanciación de una variable de la regla original.

Consideremos una regla de centro expandido de la forma general: $\phi \rightarrow \psi / \lambda \_ \rho$ y sea I el conjunto de posibles instanciaciones de valores para las variables de rasgos que esta contiene. Luego la colección de reglas instanciadas es simplemente:

$$\phi\,[i] \rightarrow \psi\,[i] / \lambda\,[i] \_ \rho\,[i]\, i \in I$$

Los componentes de las reglas en este conjunto son lenguajes regulares sobre segmentos de símbolos no analizables, con todas las matrices de rasgos y variables resueltos. Como cada regla instanciada es formada por la aplicación de la misma substitución a cada uno de los componentes de la regla original, la correlación entre componentes de elecciones de símbolos es propiamente representada. El comportamiento de la regla original es luego modelada por la relación que corresponde a

cada aplicación del batch de la elección de símbolos es propiamente representada. El comportamiento de la regla original es luego modelada por la relación que corresponde a la aplicación del batch de reglas de este conjunto, y ya se ha probado que tal relación es regular.

1.5. Las relaciones regulares como conjuntos de strings input/output de gramáticas con reglas sensibles al contexto no cíclicas

De todo lo anterior se tiene que:

Teorema 1 *Si $G = \{R_1, \ldots, R_n\}$ es un gramática definida por una secuencia totalmente ordenada de reglas de reescritura, cada una de las cuales denota una relación regular, entonces el conjunto de pares de strings input-output para esta gramática como un todo es la relacion regular dada por $R_1 \circ \ldots \circ R_n$.*

Corolario 1 *Los pares de strings input-output de una gramaática de reescritura como la de arriba son aceptados por un único traductor finito.*

Otras gramáticas se pueden modelar combinando relaciones regulares:

Supongamos que una gramática es especificada por un conjunto finito de reglas pero una especifición indica que en algunas subsecuencias de estas deben actuar en bloque de alternativas mutuamente exclusivas. Esto es solo un regla en cada una de tales subsecuencias puede ser aplicada en cualquier derivación y la elección de la alternativa varía libremente. La aplicación alternativa de las reglas en bloque puede ser modelada con la unión de las relaciones regulares que estas notan individualmente, y, se mantiene la regularidad, pues las relaciones regulares son cerradas bajo esta operación.

En un arreglo más complejo, la gramática puede especificar un conjunto de alternativas en bloque constituida por bloques que no son adyacentes en la secuencia de orden de aplicación. Por ejemplo supongamos que la gramática G consista de la secuencia de reglas $< R_1, R_2, R_3, R_4, R_5 >$ donde R_2 y R_4 constituyan un bloque de alternativas exclusivas. Esto no se puede modelar uniendo directamente R_2 y R_4 porque asi no se incorpora el efecto de la regla intermedia R_3. Sin embargo,

esta gramática se puede ver abreviando la elección entre dos secuencias diferentes: $< R_1, R_2, R_3, R_5 >$ y $< R_1, R_3, R_4, R_5 >$ y por lo tanto nota a la relación regular:

$$< R_1 \circ [(R_2 \circ R_3) \circ (R_4 \circ R_5)] \circ R_5 >$$

Así, la unión y concatenación de operadores puede ser interpuesta de diferentes maneras para obtener las ralaciones regulares que modelen toda gramática de reglas reescritura no cíclicas.

Sin embargo, existen reglas que no pueden ser expresadas por medio de estas combinaciones. Esto vale, por ejemplo, para la aplicación cíclica no restricta de reglas sobre una secuencia finita. Así, si consideramos la regla:

$$R : \lambda \rightarrow ab/a\_b$$

Se sabe que esta regla de reescritura no puede ser modelada por una relación regular si se le permite actuar sobre el material de una aplicación previa porque asi se podría generar el lenguaje libre de contexto $L = \{a^n b^n | n \geq 1\}$. Inclusive, en el peor caso, se sabe que el comportamiento de una máquina de Turing arbitraria puede ser simulado con replicaciones irrestrictas. Existen, sin embargo, alternativas que aseguran regularidad aún el caso de cierto tipo ciclicidad de las reglas, pero estas deben ser analizadas en cada caso y deberían de ser especificadas y tenidas en cuenta por los lingüístas en las descripciones.

Capítulo 2

El modelo de morfología de dos niveles

2.1. Introducción. El Problema de la cantidad de estados de los traductores en cascada

La idea de que las reglas fonológicas pueden ser implementadas con traductores finitos tiene su origen en el trabajos contemporáneos de Johnson [32], Heintz [26][27] y Kay y Kaplan [33]. En sus propuestas estos autores esbozan dos ideas: las reglas fonológicas pueden ser implementadas con traductores finitos y, segundo, estos traductores pueden componerse en cascada en series que simulan el accionar de una gramática con reglas de la fonología generativa.

En general, es posible implementar n reglas ordenadas teniendo n traductores finitos corriendo sobre $n+1$ cintas , de las cuales $n-1$ son intermediarias. Por eso, cuando n es grande el sistema se vuelve más pesado. Esto en el caso del análisis de palabras es particularmente cierto, pues para cualquier string de superficie para un Traductor finito puede haber un gran número de strings del lado léxico. Aparentemente, se obtendrá un un incremento en el número de cintas (exponencial en el peor caso) cuando n crece.

Existe, al menos teóricamente, una solución a este problema. Se sabe desde Schutzenberger [49] que todo sistema de TF en cascada puede ser compuesto en un solo FST con el mismo comportamiento con respecto al input/output. Pero incluso en este caso, en el peor caso, el traductor obtenido en la composición tendrá tantos estados como el producto de la cantidad de estados de las máquinas originales. A

pesar de que en muchos casos la cantidad de estados puede ser reducida por técnicas estándar de minimización (Hopcroft y Ullman[30]). Sin embargo las máquinas producidas por la composición tendrán todavía un número prohibitivo de estados y por eso, en parte, la idea de Kaplan y Kay no fue nunca exitosa en un modelo computacional real.

Por esta razón, entre otras, Koskenniemi [41] propuso que las reglas fonólogicas pueden ser implementadas por un conjunto de traductores finitos operado en paralelo más que en serie. En este sistema, todo traductor ve simultaneamete las cintas de superficie y léxica. Estas cintas son la únicas dos que usa el sistema y de allí el nombre de morfología de dos niveles (Two level morphology).

Una fuerte ventaja de la propuesta de Koskenniemi es que, a diferencia de los traductores en cascada, los traductores en paralelo no necesitan ser combinados en un sólo traductor grande para que el sistema funcione. De allí proviene la diferencia entre la complejidad de los dos sistemas. Mientras que los traductores finitos en cascada pueden llevar un crecimiento exponencial en la cantidad de cintas intermedias, los traductores en paralelo (cuando funcionan como aceptores de pares de cintas y cuando no hay inserciones o deleciones) corren en tiempo lineal. El número de reglas (llamado usualmente la constante de la gramática) funciona como un multiplicador constante, si se dobla el número de reglas, luego para toda correspondencia léxico/superficie chequeada en un par de strings input el sistema tiene que avanzar los estados para el doble de máquinas, y por tanto el tiempo total de computación para aceptar el par se dobla. Así, el sistema es aún lineal. Debe observarse que si existen deleciones o inserciones cuando las máquinas no corren solamente como aceptores la linealidad ya no se puede garantizar.

2.2. Reglas del modelo de morfología de dos niveles

En contraste con el modelo de gramática generativa, el modelo de morfología de dos niveles (DN) trata cada palabra como una correspondencia entre dos strings en dos niveles, el de representación léxica y el de representacién en superficie.

Por ejemplo consideremos la forma léxica tati y su representación de superficie taci:

```
String de superficie t a c i
                     | | | |
String del léxico    t a t i
```

Las reglas de morfología de dos niveles difieren de las generativas en los siguientes puntos (Antworth[4]):

1. Las reglas generativas se aplican generalmente en orden secuencial, las reglas de dos niveles (RDN) se aplican simultáneamente.

2. Las reglas generativas crean niveles intermedios de derivación, la aplicación simultánea de las RDN requieren solo dos niveles, uno léxico y uno de superficie.

3. Las reglas generativas relacionan los niveles subyacente y de superficie por medio de símbolos de reescritura, las RDN expresan las relaciones entre la forma subyacente y la de superfcie con correspondencias directas y estáticas entre pares de símbolos subyacentes y de superficie. Por tanto, después de que una RDN se aplica, ambas formas, de superficie y subyacente, existen.

4. Las reglas generativas tiene acceso solo al nivel vigente en cada estadio de la derivación, las RDN tienen acceso a los contextos de superficie y léxico. Las reglas generativas no pueden mirar atrás hacia los contextos subyacentes y no pueden mirar hacia adelante hacia el contexto de superficie. En contraste, los contextos de morfología de dos niveles se establecen como correpondencias de léxico/superficie. Esto significa que una RDN puede referirse, por ejemplo, a una a subyacente que corresponde a una b de superficie o a una b de superficie que corresponde a una a subyacente. En la fonología generativa la interacción entre reglas está controlada por los requerimientos que da que se apliquen en un cierto orden. En MDN las interacciones entre las reglas estan governadas más bien por un cuidadosa especificación de los contextos como strings de las correspondecias a dos niveles que por el orden.

5. Las reglas generativas son unidireccionales , es decir, actúan es sentido subyacente→superficie, las RDN son bidireccionales. Luego en el modo de generación las RDN aceptan una forma subyacente como input, devolviendo la forma de superficie, en

el modo de análisis tomando como output una forma de superficie devuelven la forma subyacente. La aplicación práctica de la bidireccionalidad de las reglas es obvia, una implementación computacional no se restringe solamente a un modo de generación para producir palabras, también puede ponerse en el modo de análisis para parsear palabras.

Una regla del modelo MDN esta compuesta por tres partes:

1. La correspondencia(Co).
2. El operador(Op).
3. Los contextos a izquierda y derecha (CxI y CxD).

La forma general de una regla de dos niveles es:

```
Co Op CxI___CxD
```

Por ejemplo para la regla de Palatalización de la *t*:

```
t:c  <=>___@:i
```

se tiene:

1. `Co t:c`
2. `Op <=>`
3. `CxD @:i`

En este ejemplo el contexto a izquierda no se nota por irrelevante.

La primer parte de una regla es la correspondencia. Esta especifica que par de carateres léxico/superficie controla la regla. Una correspondencia se escribe con la notación caracter léxico:caracter de superficie, por ejemplo t:t, a:a y t:c . En MDN debe existir un correpondencia uno a uno entre caracteres en la forma de superficie y la léxica.

Existen dos tipos de correspondencias

1. las correspondencias por default, como t:t .

2. las correspondencias especiales, como t:c.

La unión de la correspondencias especiales con las por default da el conjunto de pares posibles (feasible pairs) de la descripción.

La segunda parte de la regla es el operador ⇒ que indica la relación que existe entre la correspondencia y el contexto en que esta ocurre. En este ejemplo de la palatalización especifica que a una t léxica le corresponde una c en superficie, si y sólo si el par esta precediendo a la i de superficie.

Las descripciones con morfología de dos niveles utilizan cuatro operadores para especificar reglas:

1. el operador de reglas de Restricción: ⇒ (la correspondencia implica la presencia del contexto).

2. el operador de Coerción ⇐ (el contexto implica la presencia de la correspondencia).

3. el operador Bicondicional.⇔ (la correspondencia es permitida si y sólo si esta presente el contexto de la regla en cuestión).

4. el operador de Prohibición / ⇐ (indica que la correspondencia especificada esta prohibida en el contexto de la misma).

La tercera parte de una regla es el contexto que indica bajo que condiciones se produce la correspondencia especificada. Se utiliza el caracter :__ para separar los dos contextos: a izquierda y a derecha. Estos contextos especifican series de pares de caracteres y, además, se pueden utilizar los siguientes elementos notacionales:

1. cuando los dos caracteres de un par son iguales basta con especificar un caracter.

2. si no se especifica un caracter de un nivel se supone alguno cualquiera del alfabeto.

3. el símbolo # se usa como frontera de palabra.

4. se puede usar expresiones regulares para expresar contextos complejos.

5. el caracter % se utiliza como caracter de escape.

2.2.1. El uso del caracter 0

El caracter 0 es de fundamental importancia en MDN. Como los strings de superficie y léxico deben tener la misma cantidad de caracteres. La deleción de símbolo X se modela con la correspondencia $X : 0$ y la epéntesis de X con $0 : X$. Estos ceros son usados como un mecanismo de aplicación que existe solo internamente y por tanto no se imprimen en la output final. Como un ejemplo de epéntesis se puede considerar este correspondiente al Tagalog [4]. El Tagalog, tiene un infijo $-um-$ que se aglutina entre la primer consonante y la primer vocal de la raíz. Por ejemplo, la forma con el infijo de *bili* es *bumili*. Para representar este proceso con morfología de dos niveles se representa la forma subyacente del infijo $-um-$ como el prefijo $X+$, con X un símbolo especial que no tiene otra finalidad más que la de representar es infijo. Se escribe luego la regla que inserta la secuencia $-um-$ en presencia del prefijo $X+$-.

La correspondencia de DN es:

```
FL      X + b 0 0 i l i
FS      0 0 b u m i l i
```

La regla para la inserción es:

```
Infijación

X:0 <=>___+:0 C:C 0:u 0:m V:V
```

Aquí C y V representan cualquier elemento del subconjunto de consonantes y vocales, respectivamente. Así procesos de inserción como la geminación, reduplicación e infijación pueden ser modelados mediante strings donde el elemento léxical es 0.

2.3. Extensión de la demostración de Kaplan y Kay al modelos de dos niveles

2.3.1. Aplicación de los autómatas en paralelo

Los conjuntos de strings de las generalizaciones individuales del sistema de Koskenniemi son regulares, pues son definidos directamente a partir de traductores finitos

con los que se implementan las reglas. Sin embargo, no es inmediatamente obvio que la relación definida por toda la gramática de dos niveles sea regular. Un par de strings es generado por gramática de dos niveles si el par es aceptado independientemente por cada uno de los traductores, e inclusive, el label tomado por un TF en un posición particular del string es idéntico al de todo TF en esa posición del string. En esencia, se preescribe una función de transición δ para todo traductor de la gramática que permite transiciones entre estados en conjuntos de estados de producto cruzado sólo en el caso de que sean permitidos por las transiciones de matching-literal en las máquinas individuales.

Esta función de transición generaliza a traductores de dos cintas la construcción de AF para la intersección de dos lenguajes. Se debe pues sospechar que la relacion léxico superficie de las reglas de dos niveles para una grámatica que tiene como traductores a $T_1, \ldots, T_n$, es la relación:

$$R = \cap_i R\left(T_i\right)$$

Sin embargo, lo que realmente se computa bajo esta interpretación es la relación:

$$Rel\left(Rec\left(T_1\right) \cap Rec\left(T_2\right) \ldots Rec\left(T_n\right)\right)$$

Como ya fue observado en la sección 1.23 esta puede ser solamente un conjunto propio de la relación R si las relaciones componentes contienen pares de strings de longitud distinta. En este caso el traductor matching literal puede no aceptar la intersección, pues esta relación puede no ser regular.

Los traductores individuales que permiten las especificaciones de MDN permiten la expansión y contracción de strings mediante el uso del símbolo nulo 0. Si este es tratado sólo como λ (la palabra nula) se puede decir muy poco acerca de la relación combinada. Sin embargo, el efecto de la función de transición de matching literal de Koskenniemi es confeccionada para manejar el 0 como un símbolo ordinario de cinta, de manera tal que los traductores individuales son también libres de λ y la intersección de RRSIL es regular. El efecto de cambio de longitud de toda la gramática es entonces modelado por un mapeo de 0 a λ. Por tanto dentro del sistema se embebe la RRSIL R como un componente regular interno de una relación regular mayor que caracteriza el mapeo total de léxico a superficie:

$$Intro\left(0\right) \cap \left[\cap_i R\left(T_i\right)\right] \cap Intro\left(0\right)^{-1}$$

Esta relación expande sus strings léxicos introduciendo libremente símbolos 0. Estos son controlados junto con los otros símbolos en la intresección interna y el lado de superficie de la relación interna esta dado por la remoción de todos los 0. La relación externa entera da un modelo algebraico del método operacional de Koskenniemi para combinar traductores individuales y para interpretar el símbolo nulo. Con este análisis del MDN en término de operaciones regularmente cerradas y relaciones de la misma longitud, se muestra que las relaciones strings aceptadas por autómatas de dos niveles son de hecho regulares. Así, el sistema de dos niveles es de hecho uno de cuatro, pues la relación interna define dos niveles de representación internos que contienen 0. Finalmente solo dos niveles de representación son linguísticamente significativos. En las representaciones típicas de dos niveles las relaciones $Intro$ son codificadas simplemente en los algoritmos de intrepretación y no aparecen como traductores separados.

2.3.2. Reglas de restricción ($\Rightarrow$)

Notaremos por π al conjunto de pares posibles (feasible pairs). Los pares en π contienen todos los símbolos del alfabeto y el 0, no contienen a λ, exceptuando quizás al par $\lambda : \lambda$. Las relaciones correspondientes a todas las reglas individuales son todas subconjuntos de π^* y por tanto pertenecen a la clase de RRSIL (esto porque no aparecen λ apareados con otros simbolos). Para esta clase de relaciones tiene sentido hablar acerca de una correspondencia entre un símbolo en un par del string con el otro. Para modelar las condiciones impuestas por las reglas de restricción se usará el operador $IfS\ thenP$ definido en la sección. Este operador puede ser extendido para que se aplique también a relaciones regulares y el resultado será regular si los operandos están en una clase regular que sea cerrada bajo complementación. Por conveniencia notacional, se deja que la barra superior note la complementación relativa a π^* más que a la más usual relativa a $\Sigma^* \times \Sigma^*$:

$$IfP\ thenS\,(R_1, R_2) = \pi^* - R_1\overline{R_2} = \overline{R_1\overline{R_2}}$$

$$IfS\ thenP\,(R_1, R_2) = \pi^* - \overline{R_1}R_2 = \overline{\overline{R_1}R_2}$$

La condición para una regla de restricción simple es entonces modelada por la siguiente relación:

$$Restrict(\tau, \xi, \rho) = IfS\ thenP\left(\pi^*\xi, \tau\pi^*\right) \cap IfS\ thenP\left(\pi^*\tau, \rho\pi^*\right)$$

El primer componente asegura que τ aparezca siempre precedido por ξ y el segundo que este siempre seguido por ρ.

Una regla de restricción compuesta de la forma:

$$\tau \Rightarrow \xi^1 \_\_ \rho^1; \xi^2 \_\_ \rho^2; \ldots; \xi^n \_\_ \rho^n$$

es satisfecha por todas las instancias de τ rodeada por substrings que cumplen las condiciones de al menos alguno de los pares independientemente. Un candidato para modelar esta interpretación disjuntiva es la relación:

$$\cup_k Restrict\left(\tau, \xi^k, \rho^k\right)$$

Esto es incorrecto, porque el objetivo de la unión es demasiado amplio. Este especifica que debe existir algún k tal que *toda* ocurrencia de τ este rodeada por $\xi^k \_ \rho^k$. Sin embargo el k podría cambiar con la ocurrencia. Una mejor aproximación es dada por la relación:

$$\left[\cup_k Restrict\left(\tau, \xi^k, \rho^k\right)\right]^*$$

Así por la iteración de Kleene diferentes instancias de la regla pueden ser aplicadas en posiciones diferentes de los pares de strings. Sin embargo, esta definición tiene también un problema: la iteración hace que instancias diferentes de la regla matcheen en substrings separados y sucesivos. Esto impide que el substring de contexto de una aplicación pueda solaparse con el centro y porciones del contexto de la aplicación precedente.

Este problema puede resolverse con las técnicas de símbolos auxiliares que fueron desarrolladas para reglas de reescritura (ver 1.4.1). Se introducen los MCI y MCD: $<^k$ y $>^k$ para cada par de contexto $\xi^k \_ \rho^k$. Estos son distintos de todos los otros símbolos y como sus pares idénticos son ahora "feasible pairs", se agregan a π. Estos pares toman el lugar de la relación de contexto en cuestión en la unión iterativa:

$$\left[\cup_k Restrict\left(\tau, Id\left(<^k\right), Id\left(>^k\right)\right)\right]^*$$

Y con esto se elimina el problema del solapamiento. Se debe ahora asegurar que estos pares de marcadores aparezcan sólo si estan precedidos o seguidos por la

relación de contexto apropiada. Con m el conjunto de todos los pares de marcadores y con el suscripto indicando como siempre que el par idéntico de símbolos es ignorado, se define el operador de contexto a izquierda para dos niveles:

$$ContextIzq\left(\xi, l\right) = IfS\ thenP\left(\pi^{*}\xi_{m}, Id\left(l\right)\pi^{*}\right)$$

Asi $ContextIzq\left(\xi^{k}, <^{k}\right)$ da el requerimiento de que todo $<^{k}$ este precedido por una instancia de ξ^{k}. Este es más simple que el operador de para contexto a izquierda de las reglas de reescritura porque no toda instancia de ξ debe ser marcada, las marcadas son sólo las precedidas por τ y aquellas que son obtenidas independientemente por medio de la unión iterativa. Por eso se usa una implicación en un sentido más que un bicondicional. Como en el caso de reescritura el ignorar provee las instancias de ξ que se solapan.

El operador de marcadores de contexto a derecha puede ser definido simétricamente usando $IfP\ thenS$ o revirtiendo el operador de contextos a Izquierda:

$$ContextDer\left(\rho, r\right) = Rev\left(ContextIzq\left(Rev\left(\rho\right), r\right)\right)$$

Juntando todas estas piezas se tiene la siguiente relación que modela correctamente la interpretación de regla de restricción compuesta:

$$Intro\left(m\right)\ \circ$$
$$\left[\cup_{k} Restrict\left(\tau, \xi^{k}, \rho^{k}\right)\right]^{*} \cap \left[\cap_{k}\left[ContextIzq\left(\xi^{k}, <^{k}\right) \cap ContextIzq\left(\rho^{k}, >^{k}\right)\right]\right]$$
$$Intro\left(m\right)^{-1}$$

Los marcadores auxiliares son libremente introducidos en el string léxico. Estas marcas están apropiadamente ubicadas de manera tal que rodeen toda ocurrencia de τ y cada marcador marque una ocurrencia de la relación de contexto asociada. Los marcadores son removidos al final. Notemos que sólo hay relaciones entre strings de la misma longitud en la expesiones intermedias y que todos los marcadores introducidos al principio son removidos al final. Por tanto, la relación de la composición es regular y también pertenece a la subclase de relaciones con pares de strings de igual longitud. Así, el resultado de intersecar estas relaciones con pares de strings de igual longitud será también regular.

2.3.3. Reglas de coerción ($\Leftarrow$)

Una regla de coerción de la forma:

$$\tau \Leftarrow \xi \_ \rho$$

impone el requerimiento en los strings apareados que vienen entre todos los miembros de las relaciones ξ y ρ. Si el lado léxico de tales strings apareados pertenece al dominio de τ luego el lado de superficie debe ser tal que el par interviniente pertenezca a τ. Para formalizar esta interpretación se describe primero el conjunto de pares de strings que caen dentro de estas condiciones. El complemento de este conjunto es la relación apropiada.

La relación:

$$\overline{\tau} = \pi^* - \tau$$

es el conjunto de pares en π^* que no pertenecen a τ. Esto es o porque su string léxico no pertenece al dominio de τ o porque τ asocia este string léxico con strings de superficie diferentes.

Observamos que:

$$Id\left(Dom\left(\tau\right)\right) \circ \overline{\tau}$$

es el subconjunto de esos strings de superficie cuyos strings léxicos están en el dominio de de τ y cuyos strings de superficie deben por lo tanto ser diferentes de los que τ provee. Los pares de strings no aceptables deben petenecer entonces a la RRSIL $\pi^* \xi \left[Id\left(Dom\left(\tau\right)\right) \circ \overline{\tau}\right] \rho \pi^*$, su complemento regular es el operador de $Coerc$

$$Coerc\left(\tau, \xi, \rho\right) = \overline{\pi^* \xi \left[Id\left(Dom\left(\tau\right)\right) \circ \overline{\tau}\right] \rho \pi^*}$$

Este operador, por definición, contiene todos los pares de strings que satisfacen la regla.

Para la mayoría de las coerciones de superficie se tiene el caso en que estas contengan sólo los pares que satisfacen la regla. Pero para un tipo especial de coerciones, las reglas de epéntesis esta relación incluye más pares que los que se desean. Estas reglas en la cuales el dominio τ incluye strings consistentes enteramente de símbolos 0. La dificultad proviene de la naturaleza dual de los 0 en las RDN. Se comportan

formalmente propiamente como símbolos en strings de RRSIL, pero sin embargo se los hace actuar también como el string vacío. En este sentido son similares a los λ en los centros de las reglas de reescritura y deben ser también modelados con técnicas especiales.

Se usa regla de epéntesis:

$$0 : b \Leftarrow c : c\_\_d : d$$

para ilustrar características importantes.

Si esta es la única regla en la gramática, entonces la gramática debe permitir que el par de strings $< cd, cbd >$ pero no el par $< cd, ced >$ en el cual la e aparece en vez de la b entre los símbolos c y d a nivel de superficie. Esta gramática también debe prohibir el par $< cd, cd >$ en el cual c y d son adyacentes en ambos niveles y no ocurre ninguna epéntesis. Esto es consistente con la intuición de que el 0 esta en una regla por la ausencia de algún material explícito a nivel lexical y que por lo tanto la regla debe forzar una b de superficie cuando c y d léxicas son adyacentes. En nuestro análisis esta interpretación del 0 es expresada teniendo la relación $Intro$ introduciendo libremente ceros entre símbolos cualesquiera, imitando el hecho de que λ puede ser considerada como apareciendo libremente en cualquier lado.

El par $< cd, cbd >$ está permitido como la composición de pares $< cd, c0d >$ y $< c0d, cbd >$, el primer par pertenece a la relación $Intro$ y el segundo es dado por la regla en cuestión. Pero, como los 0 son introducidos libremente, la relación $Intro$ incluye también el par idéntico $< cd, cd >$ también. La relación $Coerc$ como fue definida arriba también contiene el par $< cd, cd >(=< c\lambda d, c\lambda d >)$, pues $\lambda : \lambda$ no está en $\left[0 : 0 \circ \overline{0 : b}\right]$. La gramática como un todo entonces permite $< cd, cd >$ como una composición indeseada.

Se pueden eliminar pares de este tipo usando una relación levemente diferente para reglas de epéntesis como esta. Se debe aún prohibir pares cuando los 0 en el dominio de τ esten apareados con strings que no están en el rango. Pero, además, se quieren prohibir pares para los cuales los strings léxicos no tienen 0 apropiados para disparar la coerciones de epéntesis de la gramática. Esto puede ser otenido mediante una versión modificada de la relación $Coerc$ que también excluya realizaciones del string vacío por algo que no este en τ. Para esto se reemplaza $Dom(\tau)$ en la expresión de la definición por:

$$Dom\left(\tau \cup \{\lambda : \lambda\}\right)$$

La literatura en MDN no es explícita en cuando un regla de epéntesis debe también rechazar strings con algún otro modo de inserción. En una visión, la regla sólo restringe la inserción de strings simples (compuestos por un solo símbolo) y por lo tanto pares como $< cd, cbbd >$ y $< cd, ceed >$ pueden ser incluídos en la relación. Este punto de vista es modelado por medio de la expresión $Dom\,(\tau \cup \{\lambda : \lambda\})$.

En otro punto de vista, la regla requiere que c y d léxicas adyacentes deban estar separadas exactametne por una b en la superficie, de manera tal que $< cd, cbbd >$ y $< cd, ceed >$ deben ser también excluídos de la relación. Esta interpretación se logra utilizando 0^* en vez de $Dom\,(\tau \cup \{\lambda : \lambda\})$. No es claro cual de estas interpretaciones llevan a un formalismo más conveniente, pero cada una de ellas puede ser modelada mediante dispositivos regulares.

Karttunen y Beesley (1992,p.22) discuten una particularidad un poco diferente que aparece en el análisis de una regla de epéntesis donde un contexto es omitido (o, equivalentemente, un contexto incluye el par $\lambda : \lambda$). La regla:

$$0 : b \Leftarrow c : c\_\_d : d$$

requiere que una b que no corresponde a nada en el string léxico debe aparecer en el string de superficie después de todo par $c : c$. Si se usaron tanto las expresiones 0^* o el $Dom\,(\tau \cup \{\lambda : \lambda\})$ para definir la relación de coerción de esta regla, el efecto no será el deseado. La relación resultante no permite strings en las cuales $\lambda : \lambda$ siga a $c : c$, esto porque λ esta incluído en la expresión restrictiva del dominio. Pero $\lambda : \lambda$ sigue y precede a todo par de símbolos, y por tanto el resultado de es una relación que simplemente prohibe todas las ocurrencias de $c : c$. Si, sin embargo, se vuelve a usar el dominio sin la unión $\{\lambda : \lambda\}$ se vuelve nuevamente a la dificultad que se ha visto para RDN para epéntesis: la relación resultante asegura propiamente que nada más que b pueda ser insertado después de $c : c$, pero esto deja abierta la posibilidad de que $c : c$ no esté seguida por ninguna inserción.

Se necesita una tercera fomulación para modelar la interpretación pretendida de reglas de epéntesis de un contexto. Esta es dada para contextos a la izquierda, sin contextos a la derecha especificados, por la relación:

$$\overline{\pi^* \xi \overline{\tau} \pi^*}$$

Para contextos análogos a la derecha la relación correspondiente es:

$$\overline{\pi^* \overline{\tau} \rho \pi^*}$$

Estas relaciones excluyen todos los strings donde una instancia del contexto relevante no este seguido por pares de strings que pertenezcan a τ, o porque el número apropiado de 0s no fue introducido o porque estos 0s corresponden a material de superficie no aceptable. Estas dos prescripciones pueden juntarse en una sola fórmula para todas las reglas de un solo contexto:

$$R = \overline{\pi^* \xi \overline{\tau} \rho \pi^*}$$

donde cuando un contexto esta ausente se trata como el par identidad $\lambda : \lambda$.

Nótese la similaridad entre esta fórmula y la original de la relación *Coerc*. Si se considera que:

$$\overline{\tau} = Id\left(Dom\left(\pi^*\right)\right) \circ \overline{\tau}$$

entonces:

$$R = \overline{\pi^* \xi \left[Id\left(Dom\left(\pi^*\right)\right) \circ \overline{\tau}\right] \rho \pi^*}$$

La definición general del la relación *Coerce* que modela las coerciones sean estas epentéticas o no es:

$$Coerc\left(\tau, \xi, \rho\right) = \overline{\pi^* \xi \left[Id\left(Dom\left(X\right)\right) \circ \overline{\tau}\right] \rho \pi^*},$$

donde:

$X = \tau$ si τ no tiene pares epentéticos.

$X = \tau \cup \{\lambda : \lambda\}$(o quizás $[0:0]^*$) si τ tiene solo pares epentéticos y ni ξ ni ρ contienen $\lambda : \lambda$.

$X = \pi^*$ si τ tiene sólo pares epentéticos y ξ o ρ contienen $\lambda : \lambda$

Esta definición asume que τ es homogéneo en el sentido de que todos sus pares de strings son epentéticos o bien ninguno, pero se debe analizar más para garantizar que ese sea el caso. En el formalismo que que estamos considerando, τ puede ser una RRSIL arbitraria, y no sólo el tópico par de longitud uno que aparece en RDN. Si τ contiene más de un par de strings la regla individual es interpretada como inponiendo las restricciones que deberían ser impuestas por una conjunción de reglas fomadas al substituir por τ cada miembro de los pares de strings en turno. Sin

mayor especificación e inclusive si τ contiene una cantidad infinita de pares, esta es la interpretación modelada por el relación $Coerc$, suponiendo que τ es homogéneo.

Para trabajar con un string τ no homogéneo, se separa la parte epentética y la no epentética en dos subrelaciones distintas y homogéneas. Dada un relación arbitraria τ se realiza la partición en las subrelaciones homogéneas : τ^0 y $\tau^{\overline{0}}$

$$\tau^0 = Id(0^*) \circ \tau$$
$$\tau^{\overline{0}} = \tau - \tau^0$$

Se puede reformular una regla de la forma $\tau \Leftarrow \xi\_\rho$ como la conjunción de las dos reglas siguientes:

$$\tau^0 \Leftarrow \xi\_\rho$$
$$\tau^{\overline{0}} \Leftarrow \xi\_\rho$$

Estas reglas trabajando juntas dan la interpretación deseada de la original, y cada una de ellas es apropiadamente modelada por exactamente una de las variantes de la relación $Coerc$. Una vez que se abordaron las complejidades mayores que las reglas de coerción de superficie presentan, las fomas compuestas de estas reglas son muy fáciles de modelar. Una regla de la forma:

$$\tau \Leftarrow \xi^1 \_\_ \rho^1; \xi^w \_\rho^2; \ldots; \xi^n \_\_ \rho^n$$

es intrepretada como coercionando el lado de superficie de τ si alguna de las condiciones se encuentran. Los símbolos auxiliares no son necesarios para modelar esta interpretación, pues no hay iteración como para que aparezcan la dificultades que aparecen en los solapamientos. La relación para esta regla será entonces simplemente la intersección de las relaciones que modelan las reglas componentes:

$$\cap_k Coerc\left(\tau, \xi^k, \rho^k\right)$$

2.3.4. Reglas de prohibición

Consideremos la regla de prohibición:

$$\tau/ \Leftarrow \xi\_\rho$$

Esta regla indica que un substring apareado no debe pertenencer a τ si aparece entre instancias de ξ y ρ y su lado léxico en el dominio de τ. Se puede contruir una regla de coerción de superficie estándar que tenga exactamente esta interpretación. En efecto, usando el complemento de τ restringido al dominio de τ mismo, se tiene:

$$[Id\,(Dom\,(\tau)) \circ \overline{\tau}] \Leftarrow \xi \_\_ \rho$$

Como es requerido, el lado izquierdo es la relación que mapea cada string en el dominio de τ a todos los otros strings, diferentes de los cuales τ mapea. Así, las relaciones de prohibición se reducen a coerciones de superficie ordinarias.

2.3.5. Gramáticas con reglas de dos niveles

La relación para gramáticas de reglas de dos niveles es formada tal cual la de traductores en paralelo. La intersección de las relaciones para todas las reglas individuales es construída por RRSIL internas. Esta es luego computada con la introducción de 0s y con la relaciones de *Removal* para formar un mapa externo léxico-superficie. Por lo tanto las gramáticas basadas en RDN denotan relaciones regulares.

Algunas gramt́icas hacen uso de reglas con marcadores de límites de morfemas, en este caso un símbolo especial # puede aparecer en contextos para marcar el comienzo o el final de los strings. Esto se modela con la misma técnica que se ha esbozado para reglas de reescritura; se compone al principio y al final de la cascada de cuatro niveles la relación adicional:

$$\left[\lambda : \#\ Id\left(\overline{\Sigma^*_{m0}\#\Sigma^*_{m0}}\right) \lambda : \#\right]$$

Ritchie (1992) demostró que las gramáticas de dos niveles con reglas con marcadores de borde de morfemas son completas para las gramáticas regulares.

Es importante tener en claro en estos análisis la distinción entre reglas internas y externas. Ritchie (1992), por ejemplo, probó que lenguajes regulares generados por RDN con contextos regulares son cerradas con respecto a la intersección, pero este resultado no vale si la gramática del lenguaje es tomada como una relación externa. Se definen las gramáticas G_1 y G_2 con únicas reglas, R_1y R_2, dadas por:

G_1: $\{(a:b),(0:c)\}$
R1 : $a:b \Rightarrow \_\_$

G_2: $\{(a:c),(0:b)\}$
R2 : $a:c \Rightarrow \_\_$

El dominio de ambas relaciones externas es a^*. Un string a^n es mapeado por G_1 en strings conteniendo n símbolos b con símbolos cs libremente entremezclados. Análogamente a^n es mapeado por G_2 en strings conteniendo n símbolos c con bs libremente entremezcladas.

El rango de la intersección de las relaciones externas para G_1 y G_2 contendrá entonces el mismo número de símbolos b y c. Este conjunto es no regular pues su intersección con el lenguaje b^*c^* produce el lenguaje libre de contexto b^nc^n. La intersección de dos relaciones externas es no regular y por lo tanto no puede ser la relación externa de cualquier gramática regular de dos niveles.

2.4. Implementación del modelo de dos niveles. El sistema KIMMO

A la brevedad de que apareciera la disertación de Koskenniemi, Lauri Karttunen y otros produjeron una implementación en LISP del modelo de morfología de dos niveles al que denominaron KIMMO (Kartunnen 1983). Los componentes principales del parser de KIMMO se muestran en la figura . Este programa tiene dos componentes de análisis principales: El componente de reglas y el lexicón. El primer componente contiene las reglas de dos niveles. El segundo, lexicón, contiene todos los morfemas es su forma léxica junto con las reglas de formación de palabras. Usan estos dos componentes las funciones de análisis (Recognizer) y generación (Generator). En la figura 2.1 se ilustra el funcionamiento de estas dos funciones utilizando ambos componentes:

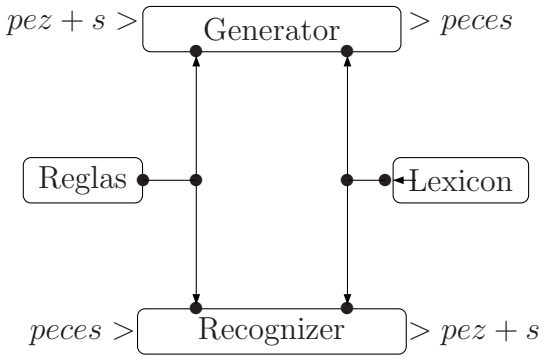


Figura 2.1: Esquema de KIMMO

En 1990 el SIL (Summer Institute of Linguistics) produjo el parser PC-KIMMO en su version 1, una implementación del modelo de morfología de dos niveles que sigue fielmente la de Karttunen. Escrito en C, este programa corría sobre sobre PCs IBM compatibles, sobre Macintosh y sobre UNIX. A pesar de ser un buen parser, este programa adolecía de la incapacidad para identificar clases de palabras. Así, por ejemplo, para la entrada *alargamientos* devolvería $alarga - miento - s$ pero no podría identificarla como un sustenativo masculino plural. Este problema se subsanó en la segunda versión de 1993 incorporando un tercer componente, una gramática de palabras.

2.4.1. El componente de traductores finitos de KIMMO

Con los traductores finitos se implementan las reglas de dos niveles. Más específicamente se utilizan máquinas de estados finitos de dos cabezas que se mueven conjuntamente en los strings léxicos y de superficie de la correspondencia. Por ejemplo el traductor T que modela la correspondencia:

Ejemplo 2.1 .

```
t:c <=>__i
```

es:

```
          t t i @
          c @ @ s
state 1: 3 2 1 1
state 2: 3 2 0 1
state 3. 0 0 1 0
```

En esta notación `@` es el caracter comodín.

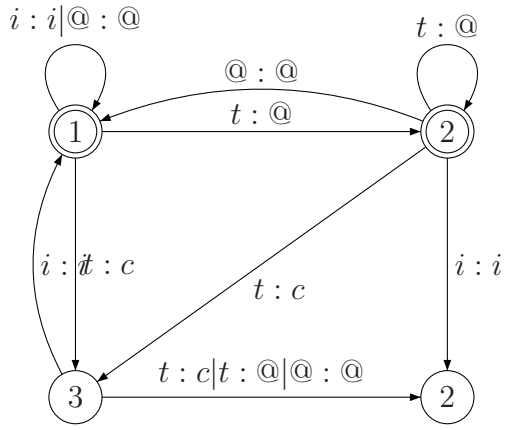


Figura 2.2: Diagrama de estados del traductor de 2.1

Un lenguaje exhibe generalmente varios procesos fonológicos, pero ejemplo Karttunen menciona que el análisis de Koskenniemi del finlandés usa 21 reglas.

De todos modos estos procesos pueden por separado ser codificados por medio de autómatas con el sistema KIMMO. En el procesamiento en curso, los autómatas pueden expresar varias reglas fonológicas en paralelo.

Como ya se mencionó, una correspondencia es aceptada si y sólo si todo traductor la acepta. Como los traductores están conectados en paralelo más que en serie no existe feeding entre los traductores de dos niveles.

2.4.2. El Lexicón de KIMMO

El diccionario de KIMMO está dividido en secciones llamadas lexicons, las cuales son finalmente alcanzables desde un lexicón de raíces. En el nivel de procesamiento del diccionario de palabras como: *perros*, KIMMO ubica en primer lugar la forma *perro* en el diccionario de raíces. El mecanismo para indicar resticciones co-ocurrencia involucra un conjunto de lexicones de continuación para cada entrada y en este caso una posibilidad es $+s$. En la forma de operar actual del sistema KIMMO el procesamiento del diccionario es llevado a cabo eficientemente por autómatas de tal manera que los dos componentes (el de reglas y el diccionario) se restringen mutuamente.

2.5. Descripción de dos niveles de la morfosintaxis verbal de quichua santiagueño

Introducción

El quechua es considerado como una familia de lenguas debido a que las variedades regionales difieren suficientemente como para no ser considerados dialectos. En los países andinos, Ecuador, Perú y Bolivia el quechua es la principal lengua aborígen. La cantidad de hablantes está estimada entre 8.5 y 10 millones. Su dispersión, aunque no abarca una zona continua desde Sur de Colombia hasta la Provincia de Santiago del estero en Argentina. Fue el lenguaje administrativo del imperio Inca. Tipológicamente, el quechua es una lengua aglutinante. Su estructura está casi enteramente basada en el uso de sufijos y es extremadamente regular. Las alteraciones vocálicas están limitadas a una parte de los dialectos. No existen prefijos y los compuestos son excepcionales. Es una lengua de tipo nominativo-acusativo, con las marcas de caso realizadas por una clase especial de sufijos que se adjuntan al final de la frase nominal. La morfología juega un papel dominante en esta lengua. Muchas funciones que son asignadas en otras lenguas a la entonación, orden de palabras o expresiones léxicas tiene en el quechua marcación morfológica, por ejemplo está muy extendido el uso de sufijos validativos. Las características enunciadas anteriormente (uso exclusivo de sufijos, productividad y regularidad de los procesos morfológicos y pocas alteraciones fonológicas) hacen que resulte natural una descripción de la morfología de esta lengua utilizando el sistema KIMMO.

Los scripts y la base que hemos desarrollado a partir del vocabulario de de Alderete [2] pertenecen al dialecto que es hablado en la provincia de Santiago del Estero, República Argentina. El quichua de Santiago del Estero pertenece, según la clasificación de los dialectos de Alfredo Torero. al grupo QII C. Este grupo esta integrado además por los dialectos cuzqueño-boliviano, el de Ayacucho. Algunos autores como Domingo Bravo ([9]) consideran que el dialecto fué introducido en tiempos post-hispánicos de la mano de los indios que acompañaban a los conquistadores.

Fonología

En nuestra formulación de la morfología verbal finita del quichua la representación de los fonemas de Alderetes [3]:

	Ocl. sordas	Ocl. sonoras	Fr. sordas	Fr. sonoras	Nasales	Laterales	vibr. simple	vibr. mult.	semicon
Bilabial	p	b			m				w
Labiodental			f						
Dento-alveolar	t	d	s		n	l	r	rr	
Palatal	ch		sh	ll	ñ				y
Velar	k	g	h						
Postvelar	q			gg					

Cuadro 2.1: Cuadro de la consonantes del quichua santiagueño

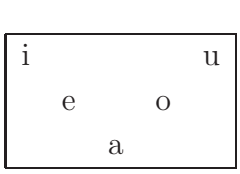


Figura 2.3: Cuadro vocálico del quichua santiagueño

Estos fonemas se introducen a nivel de la implementaciòn en el archivo quichua.RUL (A.1). Los procesos fonológicos a modelar dentro de la morfología verbal son muy escasos y tienen influencia a corta distancia:

Medialización de vocales altas

Según Nardi[1] en las palabras de origen quichua sólo se emplea e y o inmediatamente antes o después de posvelares (q y gg) y antes de los grupos consonánticos $-nq$, $-rq$ y $-yq$. Para modelar este proceso hemos definido en la implementación dos subsets (quichua.RUL, A.1):

1. Consonantes Posvelares, CPos, integrado, en primera instancia, por las consonates q y gg.

2. Fonemas transparentes, Ftr, formado por dos consontantes y una semiconsonante: y,n,r.

Se definen dos reglas de dos niveles para cada vocal que modelan la medialización cada uno de los sentidos que se implementan por medio de traductores en el archivo quichua.RUL (A.1).

R1: $i : i/ \Leftarrow$ ___ Ftr:@ CPos:@
R2: $i : i/ \Leftarrow$ CPos:@___

Los diagramas de transición de los traductores que modelan estas reglas son:

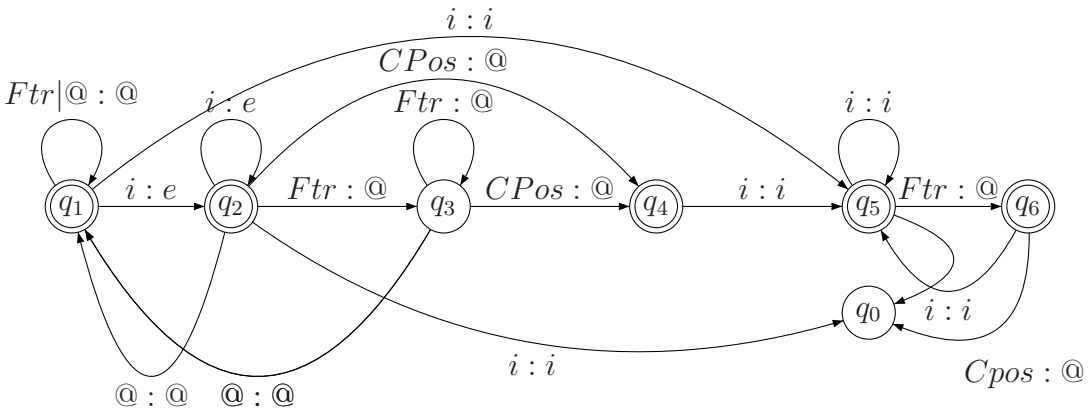


Figura 2.4: R1: $i : i/ \Leftarrow$ __ Ftr:@ CPos:@

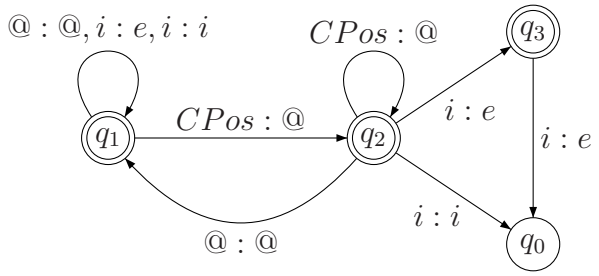


Figura 2.5: R2: $i : i/ \Leftarrow$ CPos:@__

Finalmente existe otro ámbito donde se produce otro proceso, es ante la aglutinación del sufijo que marca pasado ($-ra$). Este sufijo proviene diacrónicamente del sufijo $-rqa$, todavía presente en algunos dialectos. Como uno de los objetivos que tiene nuestra descripición es permitir una posible comparación con otros dialectos de la misma lengua se definirá una forma subyacente $-rQa$, de manera tal que se aplique la regla de medialización en sentido retrógrado. Se agrega también un par entre los pares posibles que da 0 a nivel de superficie para Q, obteniendose así la forma de superficie del sufijo que se presenta actualemente en nuestro dialecto.

Elisión y epéntesis de vocales en la aglutinación de sufijos

Algunos sufijos presentan varios alomorfos cuya presencia está determinada por el contexto. La marca de primera persona objeto se aglutina como $-wa$ luego de la secuencia aa y como a en cualquier otro contexto. Para modelar este tipo de procesos marcamos en primer lugar al fonema que se elidirá utilizando mayúsculas. En el caso del sufijo en cuestión la marca subyacente sería $-Wa$. En segundo lugar definimos una regla que impone la epéntesis (W:w) si y sólo si se presenta el contexto especificado.

R7: $W : w \Leftrightarrow a : a\ a : a + : 0\_\_$ a:a +:0

El traductor que implementa esta regla es:

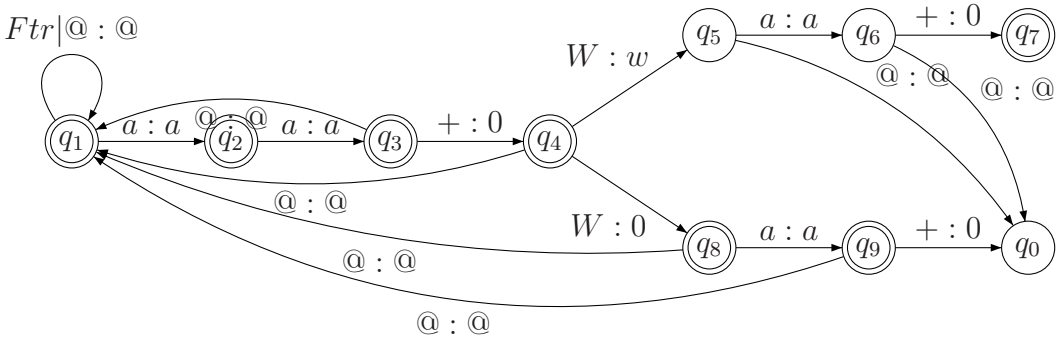


Figura 2.6: Esquema de R7: $W : w \Leftrightarrow a : a\ a : a + : 0\_\_$ a:a +:0

El resto de las reglas fonológicas suficientes para implementar la morfoligía verbal del quichua santiagueño son:

RULE 8 Elisión de la U en el cislocativo -mU:

$U : 0 \Leftrightarrow \_\_ + : 0\ p : p\ u : u + : 0$

RULE 9 Elisión de la -n en la marca de tercera pesona despues del sufijo de pasado:

$N : 0 \Leftrightarrow \_\_ + : 0\ r : r\ a : a + : 0$

Morfología

La descripción que seguiremos aquí de la morfología de las formas verbales finitas proviene de Nardi[1] y Alderetes [3].

El primer box detrás de la base esta ocupado por los sufijos de derivación verbal, entre ellos el causativo -chi que se puede aplicar hasta dos veces.

El segundo box está ocupado por las transiciones que marcan la persona objeto. Estos se pueden pluralizar pero siguiendo cierta restricciones que fueron tomadas en cuenta en la elaboración de la gramática.

El tercer box es ocupado por otros sufijos modales y el cuarto por la marca de pasado precediendo a las marcas persona de tiempo no maracado (presente), o directamente por esta personas o por las marcas de persona actor de futuro. En la confección de la gramática hemos tenido en cuenta las restricciones impuestas por la marca de persona objeto a la presencia de un marca de persona actor (Ver figura 2.4).

Por último siguen las marcas de condicional que puede estar seguida por sufijos generales como los validativos y finalmente el topicalizador (-qa) o la marca de yuxtaposición (-lla). Se modela mediante un recorrido diferente la forma de marca de segunda persona objeto que sigue a las marcas de tiempo.

En la figura 2.4 se da una descripción esquematica de la morfología en términos de autómatas finitos y en la tabla siguiente se describen las abreviaturas utilizadas. La implementación para KIMMO que representa este sistema se encuentra en el Apéndice, en el archivo quichua.grm mediante un conjunto (finito) de reglas libres de contexto.

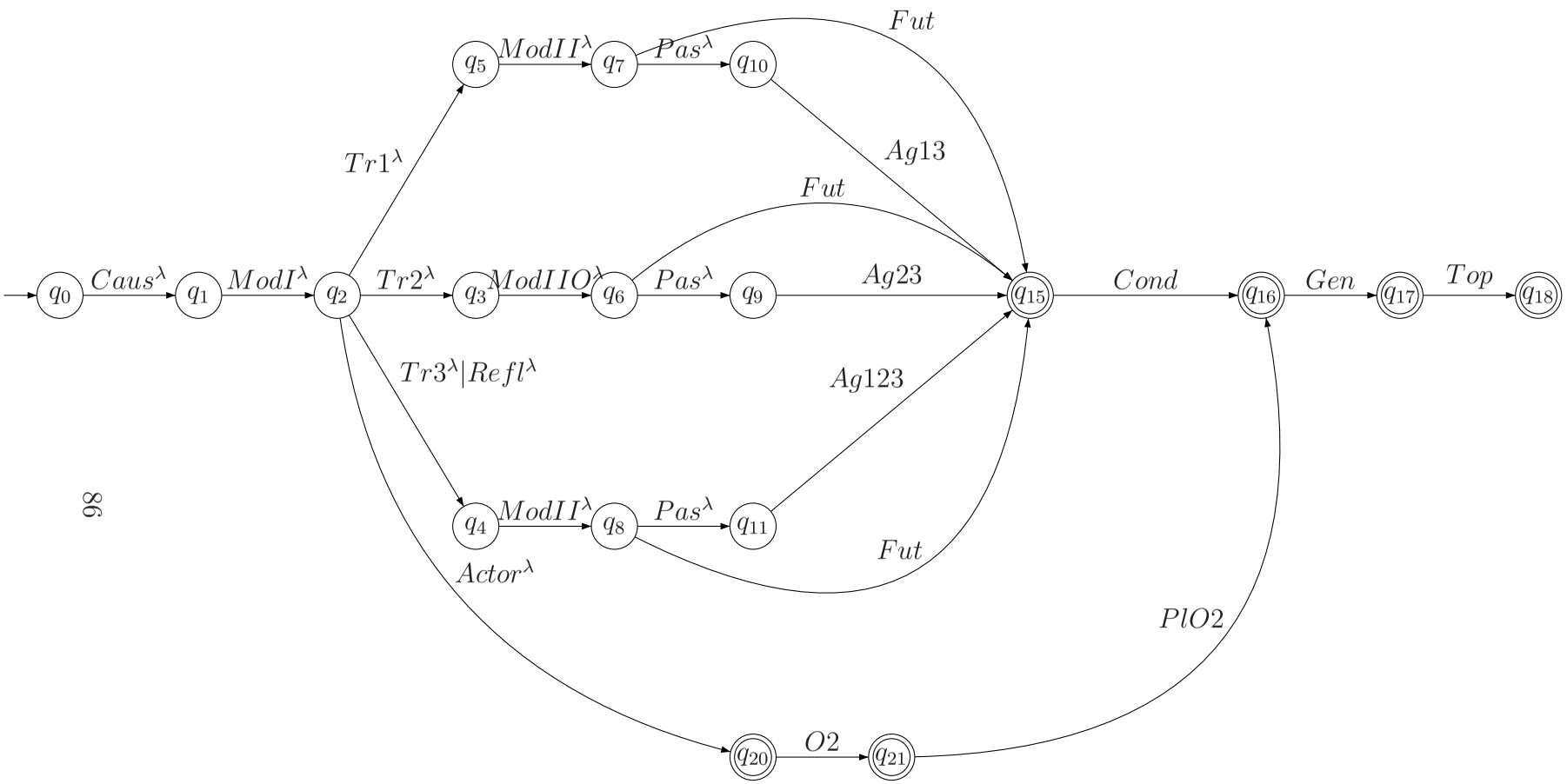


Figura 2.7: Esquema de la morfología verbal del quichua santiagueño. Los supraíndices λ indican posibles transiciones vacías

Caus.	Sufijos derivativos	-cha ≈ -ncha -ya /-a -lli
Mod I	Sufijos modales	-pa ≈ -p -ri -ia ≈ -ea -na -chi -mu ≈ -m -ku
PO	Pesona objeto	-pu -p -wa ≈-a -su
Mod II	Sufijo modal II	-naya/ -naa/ -na -chka
Pas y Fut	Sufijos temporales	-ra -sqa -sa -sajj -sqa -n-sqa
SN	Sufijos Nominalizadores	
Ag	Sufijo personal Actor	-ni -nki chis -n -ku
O II	Objeto II	-ki -chis
Cond	Subfijo Condicional	-kara
Gen	Sufijos generales	-lla -lla -pas ≈ -pis -taj -qa -chu -mi -si / -sa -cha -chus

Capítulo 3

La complejidad del modelo de morfología de dos niveles

3.1. Introducción

En este capítulo se exponen algunos resultados sobre la complejidad de modelos de morfología por autómatas de estados finitos. Si bien existen gramáticas formales que describen la morfología de algunas lenguas en particular para las que se ha demostrado que el tiempo de generación es lineal en el tamaño del input [29], a nivel abstracto, Barton [5] y [6] prueba que tanto generación y el análisis (parsing) en KIMMO son NP hard. Se expone también la discusión de Church y Koskenniemi [42] acerca de estos últimos resultados.

3.2. La Descripción del Turco y del Tatar de Heintz y Schönig

Joos Heintz presentó en 1984 [27] y en 1988 [28] una descripción de las morfologías de las lenguas Turco y Tatar en términos de lenguajes regulares . Esta descripción es independiente de los desarrollos de Kay y Kaplan a partir de los cuales Koskenniemi desarrolló el modelo de MDN. La descipción fué publicada en un trabajo conjunto con Claus Schönig en 1991 [29].

Algunas características salientes de esta descripción son:

1. Las formas subyacentes utilizan un lenguaje Regular. Los símbolos terminales de este lenguaje son las raíces y los sufijos pensados como símbolos atómicos. Además se utilizan símbolos no terminales para representar las producciones que modelan el proceso de aglutinación de subfijos. En el caso de la morfología nominal del Ottomano se marcan las raíces nominales según si terminan o no en consonante. Los símbolos no terminales que simbolizan los sufijos se notan por X_1 y X_2 para cada una de las situaciones anteriores, respectivamente. Los símbolos no terminales son:

Símbolos no terminales	Significado gramatical
Θ_1	número
Θ_2	
Π_1	posesivo
Π_2	
Π'	número del posesivo
Γ	caso

2. La aglutinación de sufijos se describe por medio de un cantidad finita de reglas libres de contexto. Por ejemplo, para la morfosintaxis nominal del Ottomano las reglas son dadas por el conjunto P_0:

$$\Theta_1 \longrightarrow [\text{LAr}]\Pi_1, \quad \Theta_1 \longrightarrow \Pi_1$$
$$\Theta_2 \longrightarrow [\text{LAr}]\Pi_2, \quad \Theta_2 \longrightarrow \Pi_2$$

$$\Pi_1 \longrightarrow [\text{Im}]\Pi', \quad \Pi_1 \longrightarrow [\text{In}]\Pi' \quad \Pi_1 \longrightarrow [\text{SIN}]\Gamma \quad \Pi_1 \longrightarrow \Gamma$$
$$\Pi_2 \longrightarrow [\text{m}]\Pi', \quad \Pi_2 \longrightarrow [\text{n}]\Pi' \quad \Pi_2 \longrightarrow [\text{SIN}]\Gamma \quad \Pi_2 \longrightarrow \Gamma$$

$$\Pi' \longrightarrow [\text{IZ}]\Gamma, \quad \Pi' \longrightarrow \Gamma$$

$$\Gamma \longrightarrow [*], \quad \Gamma \longrightarrow [\text{NIm}], \quad \Gamma \longrightarrow [\text{YI}],$$
$$\Gamma \longrightarrow [\text{YA}], \quad \Gamma \longrightarrow [\text{DA}], \quad \Gamma \longrightarrow [\text{DAN}],$$

3. Se obtiene un lenguaje Regular Monstruo a partir de traductores finitos que descoponen las formas atómicas que representan los subfijos nominales. Por ejemplo para köy[LAr][SIN][DAn] el string a nivel del lenguaje monstruo es: köyLArSINDAn.

4. Las formas de superficie son obtenidas a partir del lenguaje monstruo por traductores finitos que modelan los procesos de armonía vocálica y sandhi.

5. Se introduce el concepto de eficienciencia de la gramática:

   $$\text{eficiencia} = \frac{\text{tamaño de la gramática}}{\text{número de morfemas generados}}$$

   En el caso de la gramática que describe la morfología nominal del Ottomano dado una cantidad s de raíces el número de morfemas generados es $72s$ y el tamaño de la gramática es 72 más alguna constante. Así, cuando s es grande la eficiencia es cercana a $\frac{1}{72}$, a partir de los cual los autores consideran que la descripción es relativamente eficiente.

Con el respecto al objetivo de este apartado es importante repasar el análisis de la complejidad que hacen estos autores del problema de reconocimiento en su artículo . Utilizando que los autómatas finitos son un caso especial de máquinas de Turing de una cinta los autores pueden afirmar que el reconocimiento de la morfología del Ottomano y del Tatar es lineal en el tamaño del input según su modelado. Los autores utilizan este resultado para validar su descripción como una descripción psicólogica del procesamiento del lenguaje natural.

3.3. El análisis de Barton del modelo de Morfología de dos niveles

En esta sección, siguiendo a Barton [5] y [6], se muestra con reducciones que los autómatas de dos niveles pueden describir problemas computacionalmente complicados de una manera muy natural. Se sigue por tanto que el framework de morfología de dos niveles no garantiza per se eficiencia computacional. Si las palabras de los lenguajes naturales son fáciles de analizar es más bien debido a una propiedad adicional de las lenguas naturales que esta más allá de las capturadas por el modelo de morfología de dos niveles

3.3.1. La Generación en KIMMO es NP hard

La reducción se hará a versiones del problema de satisfacción de una fórmula boolena (SAT). Cada instancia del problema de SAT involucra una fórmula booleana en su forma normal conjuntiva (FNC) y pregunta si existe alguna opción para asignar

valores de verdad a las variables de manera tal que la fórmula sea verdadera. El problema SAT es NP-completo y por tanto computacionalmente complejo.

Es fácil codificar un problema arbitrario SAT como un problema de generación en el framework de KIMMO. El problema general de mapear de formas léxicas sería por tanto NP-hard, esto es NP-completo o peor.

Definamos una instacia del problema de generación en KIMMO como un par $< A, \sigma >$ tal que para algún σ' el par léxico-superficie σ/σ' satisface las restricciones impuestas por el autómata A. Luego $< A, \sigma >$ es una instancia de la generación en KIMMO si existe algún string en la superficie que pueda ser generado desde la forma léxica σ con el autómata A (con esta definición del problema no se necesita un algoritmo para exhibir los strings de superficie que se pueden generar, sólo decir que si existe alguno).

La codificación de un problema de SAT, f, como un par $< A, \sigma >$ se hace en dos pasos:

Paso 1.

Se construye σ desde la fórmula de f con una traslación notacional. El signo menos se usa para negación, la coma para la conjunción y no se explicita la disjunción. Por ejemplo el σ correspondiente a la fórmula:

$$(\bar{x} \vee y) \wedge (\bar{y} \vee z) \wedge (x \vee y \vee z)$$

es:

$$-xy, -yz, -xyz$$

Obsérvese que la notación no es ambigua a pesar de no tener paréntesis porque está en su FNC.

Paso 2

Se construye el traductor en tres partes (A varía de fórmula en fórmula sólo cuando las fórmulas involucran diferentes conjuntos de variables). La especificación del alfabeto en KIMMO debe listar las variables en σ y los caracteres especiales T,F, el signo menos y la coma. El signo igual debe ser declarado como el comodín.

Se definen diversos traductores, uno por variable, que verifican consitencia:

```
"consistencia de u" 3 3
    u  u  @
    T  F  @

 1: 2  3  1
 2: 2  0  2   u verdadero
 3: 0  3  3   u falso
```

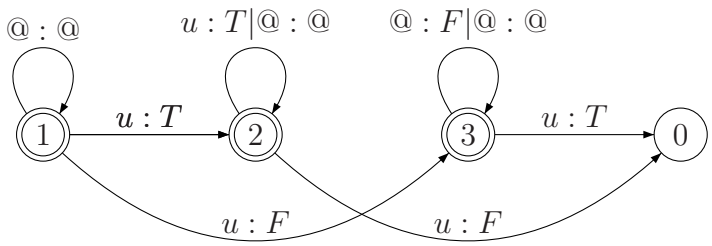


Figura 3.1: Traductor de consistencia

Cada autómata de consistencia recuerda el valor de verdad para una variable y por tanto se necesita un autómata diferente para cada variable en σ.

Se define, también el traductor de satisfacción:

```
"satisfacción" 3 4


         @ @ - ,
         T F - ,
state 1. 2 1 3 0    (no hay verdadero en el grupo)
state 2: 2 2 2 1    (hay verdadero en el grupo)
state 3. 1 2 0 0    (-F cuenta como verdadero)
```

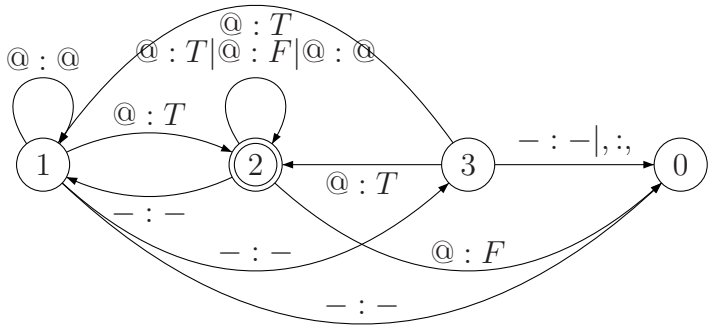


Figura 3.2: Traductor de satisfacción

Por ejemplo el sistema completo que codifica fórmulas f que usan las variables x, y y z es:

```
ALPHABET x y z T F - ,
ANY =
END
"consistencia de x" 3 3

    x  x  =
    T  F  =
 1: 2  3  1
 2: 2  0  2
 3: 0  3  3


"consistencia de y" 3 3

    y  y  =
    T  F  =
 1: 2  3  1
 2: 2  0  2
 3: 0  3  3

"consistencia de z" 3 3
```

```
   z  z  =
   T  F  =
1: 2  3  1
2: 2  0  2
3: 0  3  3

"satisfaccion" 3 4
  = = - ,
  T F - ,
1.2 1 3 0
2:2 2 2 1
1.1 2 0 0

END
```

El sistema de generación usado en esta construcción se establece de manera tal que los strings de superficie sean idénticos a los léxicos, pero con los valores de verdad substituidos en las variables. Luego, todo string de superficie generado por σ va a exhibir directamente un asignamiento de valor de verdad para f. El autómata de consistencia para cada variable u asegura que el valor asignado a la variable u es consistente en todo el string.

En el estado 1, ningún valor de verdad ha sido asignado y tanto u/T como u/F es aceptable. En el estado 2, u/T ha sido elegido alguna vez y entonces sólo u/T puede permitirse para otras ocurrencia de u. Similarmente, el estado 3 permite sólo u/F . Sin importar en cual estado este el traductor de u-consistencia ignorará todo para que no involucre a u, pasando por encima de las marcas de puntuación y otras variables. El traductor de satisfacción se bloquea si cualquier conjunción contiene solo F y $-T$ después de que los valores de verdad hayan sido substituidos en la variables. Terminará en un estado final sólo si los estados de verdad de las variables han sido asignados satisfaciendo toda disjunción y, por tanto, a f.

El resultado neto de las restricciones impuestas por la consistencia y el autómata de satisfacción es que algunos strings de superficie pueden ser generados por σ sólo en el caso de que la fórmula original, f, tenga un asignación satisfaciendo el valor de verdad. El par $< A, \sigma >$ puede ser construido en tiempo polinomial en la longitud

de σ, luego el SAT es reducible en tiempo polinomial a la generación en KIMMO y en el caso general la generación en KIMMO es como mínimo tan compleja como el SAT.

Técnicamente, esta reducción varía tanto con el conjunto de autómatas como con el string de entrada. Sin embargo, el conjunto de autómatas varía sólo de forma limitada y regular. Para un problema de satisfactibilidad de k variables, un sólo conjunto de $k+1$ autómatas con pocos estados es suficiente.

En la figura 3.3[Ba 87] se presenta un caso de algoritmo de generación en KIMMO para la fórmula:

$$(\bar{x} \vee y) \wedge (\bar{y} \vee z) \wedge (\bar{y} \vee \bar{z}) \wedge (x \vee y \vee z)$$

Debido a que sólo una asignación de verdad satisfacerá la fórmula, se necesita mucho backtracking para hallarla.

A pesar de que el generador encuentra eventualmente el valor correcto de asignación de verdad, sólo lo hace después de un tiempo substancial de búsqueda. Este ejemplo muesta que es la ambiguedad local más bien que la global la que causa problemas. Aquí solo un valor de verdad satisfacera la fórmula, luego el string es globlamente no ambiguo, pero la búsqueda de la elección correcta toma tiempo porque las asignaciones correctas no son obvias durante el procesamiento donde son hechas.

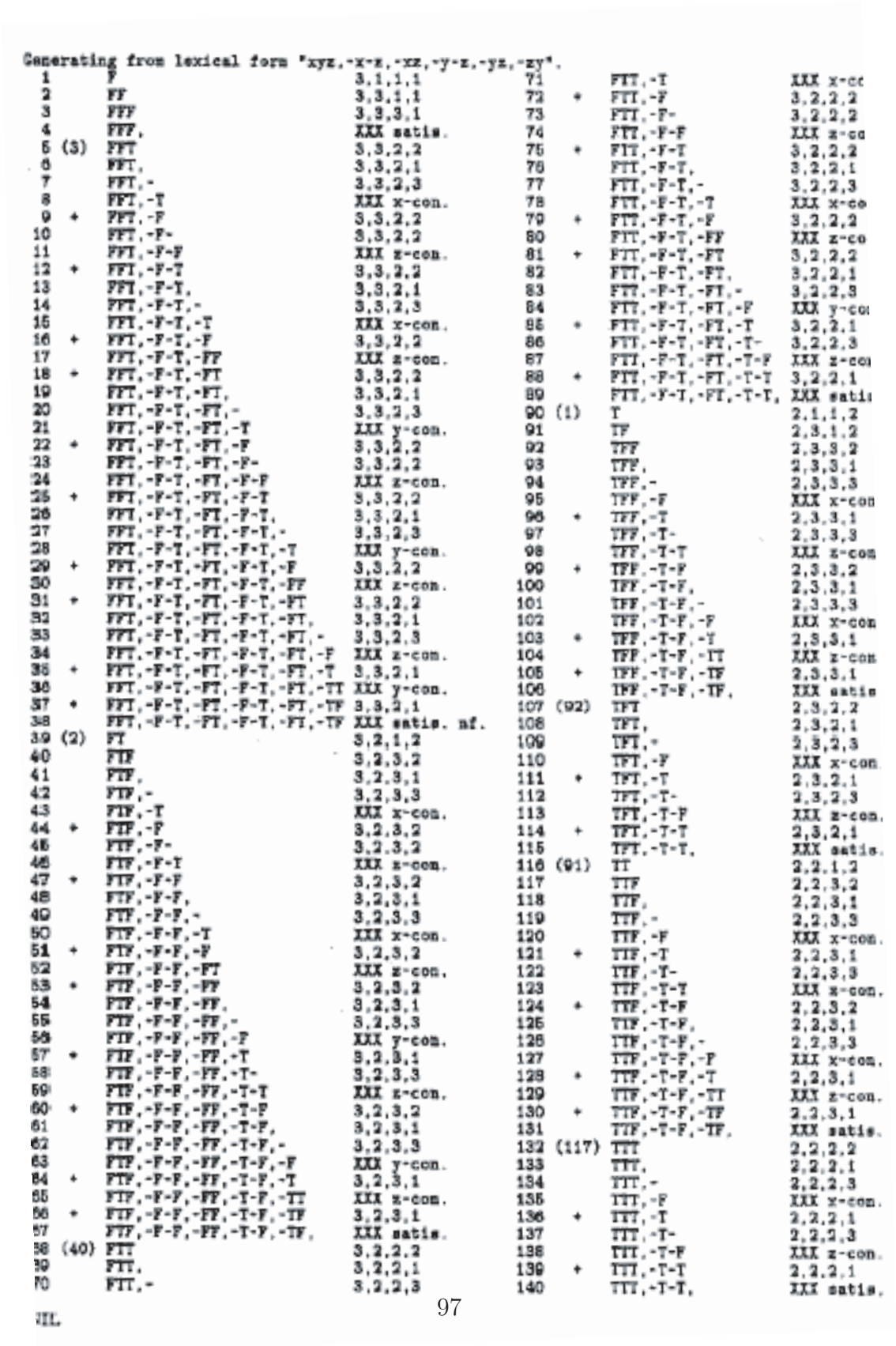
Como ejemplo de lo que sucede con un fórmula no satisfacible se muestra en la figura 3.4 el análisis de la fórmula:

$$(x \vee y \vee z) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{x} \vee z) \wedge (\bar{x} \vee \bar{y}) \wedge (\bar{x} \vee y) \wedge (\bar{y} \vee z) \wedge (\bar{y} \vee \bar{z}) \wedge (x \vee y \vee z)$$

```
Generating from lexical form "-xy,-yz,-y-z,xyz".
  1      -                  1,1,1,3          38  +    -FF,-FT,-F-T,FFT   3,3,2,2
  2      -F                 3,1,1,2          39       "-FF,-FT,-F-T,FFT" *** result
  3      -FF                3,3,1,2          40 (3)   -FT                3,2,1,2
  4      -FF,               3,3,1,1          41       -FT,               3,2,1,1
  5      -FF,-              3,3,1,3          42       -FT,-              3,2,1,3
  6      -FF,-T             XXX y-con.       43       -FT,-F             XXX y-con.
  7   +  -FF,-F             3,3,1,2          44  +    -FT,-T             3,2,1,1
  8      -FF,-FF            3,3,3,2          45       -FT,-TF            3,2,3,1
  9      -FF,-FF,           3,3,3,1          46       -FT,-TF,           XXX satis.
 10      -FF,-FF,-          3,3,3,3          47 (45)  -FT,-TT            3,2,2,2
 11      -FF,-FF,-T         XXX y-con.       48       -FT,-TT,           3,2,2,1
 12   +  -FF,-FF,-F         3,3,3,2          49       -FT,-TT,-          3,2,2,3
 13      -FF,-FF,-F-        3,3,3,2          50       -FT,-TT,-F         XXX y-con.
 14      -FF,-FF,-F-T       XXX z-con.       51  +    -FT,-TT,-T         3,2,2,1
 15   +  -FF,-FF,-F-F       3,3,3,2          52       -FT,-TT,-T-        3,2,2,3
 16      -FF,-FF,-F-F,      3,3,3,1          53       -FT,-TT,-T-F       XXX z-con.
 17      -FF,-FF,-F-F,T     XXX x-con.       54  +    -FT,-TT,-T-T       3,2,2,1
 18   +  -FF,-FF,-F-F,F     3,3,3,1          55       -FT,-TT,-T-T,      XXX satis.
 19      -FF,-FF,-F-F,FT    XXX y-con.       56 (2)   -T                 2,1,1,1
 20   +  -FF,-FF,-F-F,FF    3,3,3,1          57       -TF                2,3,1,1
 21      -FF,-FF,-F-F,FFT   XXX z-con.       58       -TF,               XXX satis.
 22   +  -FF,-FF,-F-F,FFF   3,3,3,1          59 (57)  -TT                2,2,1,2
 23      -FF,-FF,-F-F,FFF   XXX satis. nf.   60       -TT,               2,2,1,1
 24 (8)  -FF,-FT            3,3,2,2          61       -TT,-              2,2,1,3
 25      -FF,-FT,           3,3,2,1          62       -TT,-F             XXX y-con.
 26      -FF,-FT,-          3,3,2,3          63  +    -TT,-T             2,2,1,1
 27      -FF,-FT,-T         XXX y-con.       64       -TT,-TF            2,2,3,1
 28   +  -FF,-FT,-F         3,3,2,2          65       -TT,-TF,           XXX satis.
 29      -FF,-FT,-F-        3,3,2,2          66 (64)  -TT,-TT            2,2,2,2
 30      -FF,-FT,-F-F       XXX z-con.       67       -TT,-TT,           2,2,2,1
 31   +  -FF,-FT,-F-T       3,3,2,2          68       -TT,-TT,-          2,2,2,3
 32      -FF,-FT,-F-T,      3,3,2,1          69       -TT,-TT,-F         XXX y-con.
 33      -FF,-FT,-F-T,T     XXX x-con.       70  +    -TT,-TT,-T         2,2,2,1
 34   +  -FF,-FT,-F-T,F     3,3,2,1          71       -TT,-TT,-T-        2,2,2,3
 35      -FF,-FT,-F-T,FT    XXX y-con.       72       -TT,-TT,-T-F       XXX z-con.
 36   +  -FF,-FT,-F-T,FF    3,3,2,1          73  +    -TT,-TT,-T-T       2,2,2,1
 37      -FF,-FT,-F-T,FFF   XXX z-con.       74       -TT,-TT,-T-T,      XXX satis.

 ("-FF,-FT,-F-T,FFT")
```

Figura 3.3: Generación de la fórmula $(\bar{x} \vee y) \wedge (\bar{y} \vee z) \wedge (\bar{y} \vee \bar{z}) \wedge (x \vee y \vee z)$



```
Generating from lexical form 'xyz,-x-z,-xz,-y-z,-yz,-zy'.
  1       F                          3,1,1,1          71      FTT,-T              XXX x-co
  2       FF                         3,3,1,1          72   +  FTT,-F              3,2,2,2
  3       FFF                        3,3,3,1          73      FTT,-F-             3,2,2,2
  4       FFF,                       XXX satis.       74      FTT,-F-F            XXX x-co
  5 (3)   FFT                        3,3,2,2          75   +  FTT,-F-T            3,2,2,2
  6       FFT,                       3,3,2,1          76      FTT,-F-T,           3,2,2,1
  7       FFT,-                      3,3,2,3          77      FTT,-F-T,-          3,2,2,3
  8       FFT,-T                     XXX x-con.       78      FTT,-F-T,-T         XXX x-co
  9   +   FFT,-F                     3,3,2,2          79   +  FTT,-F-T,-F         3,2,2,2
 10       FFT,-F-                    3,3,2,2          80      FTT,-F-T,-FF        XXX x-co
 11       FFT,-F-F                   XXX x-con.       81   +  FTT,-F-T,-FT        3,2,2,2
 12   +   FFT,-F-T                   3,3,2,2          82      FTT,-F-T,-FT,       3,2,2,1
 13       FFT,-F-T,                  3,3,2,1          83      FTT,-F-T,-FT,-      3,2,2,3
 14       FFT,-F-T,-                 3,3,2,3          84      FTT,-F-T,-FT,-F     XXX y-co
 15       FFT,-F-T,-T                XXX x-con.       85   +  FTT,-F-T,-FT,-T     3,2,2,1
 16   +   FFT,-F-T,-F                3,3,2,2          86      FTT,-F-T,-FT,-T-    3,2,2,3
 17       FFT,-F-T,-FF               XXX x-con.       87      FTT,-F-T,-FT,-T-F   XXX x-co
 18   +   FFT,-F-T,-FT               3,3,2,2          88   +  FTT,-F-T,-FT,-T-T   3,2,2,1
 19       FFT,-F-T,-FT,              3,3,2,1          89      FTT,-F-T,-FT,-T-T,  XXX satis
 20       FFT,-F-T,-FT,-             3,3,2,3          90 (1)  T                   2,1,1,2
 21       FFT,-F-T,-FT,-T            XXX y-con.       91      TF                  2,3,1,2
 22   +   FFT,-F-T,-FT,-F            3,3,2,2          92      TFF                 2,3,3,2
 23       FFT,-F-T,-FT,-F-           3,3,2,2          93      TFF,                2,3,3,1
 24       FFT,-F-T,-FT,-F-F          XXX x-con.       94      TFF,-               2,3,3,3
 25   +   FFT,-F-T,-FT,-F-T          3,3,2,2          95      TFF,-F              XXX x-con
 26       FFT,-F-T,-FT,-F-T,         3,3,2,1          96   +  TFF,-T              2,3,3,1
 27       FFT,-F-T,-FT,-F-T,-        3,3,2,3          97      TFF,-T-             2,3,3,3
 28       FFT,-F-T,-FT,-F-T,-T       XXX y-con.       98      TFF,-T-T            XXX x-con
 29   +   FFT,-F-T,-FT,-F-T,-F       3,3,2,2          99   +  TFF,-T-F            2,3,3,2
 30       FFT,-F-T,-FT,-F-T,-FF      XXX x-con.      100      TFF,-T-F,           2,3,3,1
 31   +   FFT,-F-T,-FT,-F-T,-FT      3,3,2,2         101      TFF,-T-F,-          2,3,3,3
 32       FFT,-F-T,-FT,-F-T,-FT,     3,3,2,1         102      TFF,-T-F,-F         XXX x-con
 33       FFT,-F-T,-FT,-F-T,-FT,-    3,3,2,3         103   +  TFF,-T-F,-T         2,3,3,1
 34       FFT,-F-T,-FT,-F-T,-FT,-F   XXX x-con.      104      TFF,-T-F,-TT        XXX x-con
 35   +   FFT,-F-T,-FT,-F-T,-FT,-T   3,3,2,1         105   +  TFF,-T-F,-TF        2,3,3,1
 36       FFT,-F-T,-FT,-F-T,-FT,-TT  XXX y-con.      106      TFF,-T-F,-TF,       XXX satis
 37   +   FFT,-F-T,-FT,-F-T,-FT,-TF  3,3,2,1         107 (92) TFT                 2,3,2,2
 38       FFT,-F-T,-FT,-F-T,-FT,-TF  XXX satis. nf.  108      TFT,                2,3,2,1
 39 (2)  FT                         3,2,1,2          109      TFT,-               2,3,2,3
 40       FTF                       3,2,3,2          110      TFT,-F              XXX x-con
 41       FTF,                      3,2,3,1          111   +  TFT,-T              2,3,2,1
 42       FTF,-                     3,2,3,3          112      TFT,-T-             2,3,2,3
 43       FTF,-T                    XXX x-con.       113      TFT,-T-F            XXX x-con.
 44   +   FTF,-F                    3,2,3,2          114   +  TFT,-T-T            2,3,2,1
 45       FTF,-F-                   3,2,3,2          115      TFT,-T-T,           XXX satis.
 46       FTF,-F-T                  XXX x-con.       116 (91) TT                  2,2,1,2
 47   +   FTF,-F-F                  3,2,3,2          117      TTF                 2,2,3,2
 48       FTF,-F-F,                 3,2,3,1          118      TTF,                2,2,3,1
 49       FTF,-F-F,-                3,2,3,3          119      TTF,-               2,2,3,3
 50       FTF,-F-F,-T               XXX x-con.       120      TTF,-F              XXX x-con.
 51   +   FTF,-F-F,-F               3,2,3,2          121   +  TTF,-T              2,2,3,1
 52       FTF,-F-F,-FT              XXX x-con.       122      TTF,-T-             2,2,3,3
 53   +   FTF,-F-F,-FF              3,2,3,2          123      TTF,-T-T            XXX x-con.
 54       FTF,-F-F,-FF,             3,2,3,1          124   +  TTF,-T-F            2,2,3,2
 55       FTF,-F-F,-FF,-            3,2,3,3          125      TTF,-T-F,           2,2,3,1
 56       FTF,-F-F,-FF,-F           XXX y-con.       126      TTF,-T-F,-          2,2,3,3
 57   +   FTF,-F-F,-FF,-T           3,2,3,1          127      TTF,-T-F,-F         XXX x-con.
 58       FTF,-F-F,-FF,-T-          3,2,3,3          128   +  TTF,-T-F,-T         2,2,3,1
 59       FTF,-F-F,-FF,-T-T         XXX x-con.       129      TTF,-T-F,-TT        XXX x-con.
 60   +   FTF,-F-F,-FF,-T-F         3,2,3,2          130   +  TTF,-T-F,-TF        2,2,3,1
 61       FTF,-F-F,-FF,-T-F,        3,2,3,1          131      TTF,-T-F,-TF,       XXX satis.
 62       FTF,-F-F,-FF,-T-F,-       3,2,3,3          132 (117) TTT                2,2,2,2
 63       FTF,-F-F,-FF,-T-F,-F      XXX y-con.       133      TTT,                2,2,2,1
 64   +   FTF,-F-F,-FF,-T-F,-T      3,2,3,1          134      TTT,-               2,2,2,3
 65       FTF,-F-F,-FF,-T-F,-TT     XXX x-con.       135      TTT,-F              XXX x-con.
 66   +   FTF,-F-F,-FF,-T-F,-TF     3,2,3,1          136   +  TTT,-T              2,2,2,1
 67       FTF,-F-F,-FF,-T-F,-TF,    XXX satis.       137      TTT,-T-             2,2,2,3
 68 (40)  FTT                       3,2,2,2          138      TTT,-T-F            XXX x-con.
 69       FTT,                      3,2,2,1          139   +  TTT,-T-T            2,2,2,1
 70       FTT,-                     3,2,2,3          140      TTT,-T-T,           XXX satis.
XIL.
```


Figura 3.4: Generación a partir de $(x \vee y \vee z) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{x} \vee z) \wedge (\bar{x} \vee \bar{y}) \wedge (\bar{x} \vee y) \wedge (\bar{y} \vee z) \wedge (\bar{y} \vee \bar{z}) \wedge (x \vee y \vee z)$

3.3.2. El análisis en KIMMO es NP hard

Como el generador, un parser KIMMO puede ser usado para resolver problemas computacionalmente difíciles.

Definición 3.1 *Se define una posible instancia de un poblema computacional de análisis en KIMMO como un triplete* $< A, D, \sigma >$ *tal que para algun* σ'*:*

1. *el par léxico/superficie* σ'/σ *satisface, igual que arriba, las restricciones del autómata* A *.*

2. σ' *puede ser generado por el diccionario* D*.*

Es decir, $< A, D, \sigma >$ es una instancia del análisis en KIMMO si σ es una palabra reconocible de acuerdo a las restricciones impuestas por A y por D.

Son posibles muchas reducciones, pero la reducción que se usará aquí será el problema 3SAT en vez del SAT. Se usará también una codificación de las fórmulas FNC que es levemente diferente de la del sistema generador. Para codificar una formula SAT f como un triplete $< A, D, \sigma >$, se construye primero σ a partir de f con una nueva notación trancripcional.

Se tratará una variable u y su negación $\bar{u}$ como caracteres separados y atómicos. La coma se continuará usando para la conjunción y ningún operador explícito para la disjunción, pero añadiremos un punto al final de la fórmula.

Luego el σ correspondiente a la fórmula:

$$(\bar{x} \vee \bar{x} \vee y) \wedge (\bar{y} \vee \bar{y} \vee z) \wedge (x \vee y \vee z)$$

es:

$$x\bar{x}y, \bar{y}\bar{y}z, xyz.,$$

Un string de 12 caracteres (dentro de 3SAT las comas son redundantes, pero son retenidas para darle mas legibilidad)

Contrucción de A

Se construye A en dos partes: A, como antes, varía con las fórmulas sólo cuando las fórmulas tienen diferentes variables.

La especificación del alfabeto debe listar tanto las variables en σ como sus negaciones y los caracteres especiales: T, F, coma y punto. Nuevamente el signo @ debe declararse como el caracter comodín.

Se construye el traductor de consistencia:

```
"consistencia"' 3 5


         T T F F @
         u û u û @

state 1: 2 3 3 2 1   (u no deseado)
state 2: 2 0 0 2 2   (u verdadero)
state 3: 0 3 3 0 3   (u falso)
```

No es necesario en este caso traductor de sastisfacción. En segundo lugar, se contruye D, el lexicón:

```
   (Root=Root)
   (Punct= Punct)
   (# =)

END

LEXICON Root     TTT      Punct  "";
                 TTF      Punct  "";
                 TFT      Punct  "";
                 TFF      Punct  "";
                 FTT      Punct  "";
                 FTF      Punct  "";
                 FFT      Punct  ""

LEXICON Punct    ,        Root   "";
                 .        #      "";

END
```

En esta reducción D impone la restricción de satisfacción que fué previamente exigida por el traductor de satisfacción. La fórmula f será satisfecha si y sólo si todas sus conjunciones son satisfechas y, como f está en una forma 3CFN esto significa que los valores de verdad dentro de cada conjunción deben ser TTT, TTF,...o

cualquier combinación de los tres valores salvo FFF. Por eso, ésta es exactamente la restricción impuesta por el diccionario (obsérvese que D es la misma para todo 3SAT, que no crece con el tamaño de la fórmula o el número de variables).

Comparando con la reducción del generador, los roles de las cadenas de superficie y léxica son revertidos en el problema de análisis. El string de superficie codifica f mientras que el string léxico indica valores de verdad de las variables. El traductor de consistencia para cada variable sigue asegurando que el valor de la variable asignado a u sea consistente en toda la fórmula, pero ahora también asegura que u y $\bar{u}$ tengan valores opuestos. Como antes, el resultado neto de las restricciones hechas por los componentes en su conjunto es que $< A, D, \sigma >$ es reconocida por KIMMO sólo en el caso de que f tenga un valor de verdad que la satisfaga. El caso general de análisis en KIMMO es como mínimo tan compleja como el 3SAT, luego tan compleja como SAT o cualquier otra problema en NP (en el sentido de reducción en tiempo polinomial).

La generación y el análisis para los traductores de KIMMO sin ceros es NP-completo. Como se probó que los problemas son NP-hard, solo resta mostrar que una máquina no deterministica puede resolverlos en tiempo polinomial. Sólo se dara un sketch de la prueba.

Dada una instancia $< A, \sigma >$ de la generación en KIMMO, el no-derterminismo básico de la máquina puede usarse para preguntar por el string de superficie correspondiente al string léxico σ. El traductor puede entonces verificar rápidamente la correspondencia. Un punto crucial es que si A no permite ceros, los caracteres léxicos y de superficie deben estar en una correspondencia 1 a 1. El string de superficie debe tener la misma longitud que el string léxico y entonces el tamaño de la búsqueda no puede irse de las manos (si la búsqueda fuera muy larga la máquina puede no correr en tempo polinomial).

Dada una posible instancia $< A, D, \sigma >$ de la recognición en KIMMO, la máquina debe buscar el string léxico en vez del string de superficie, como antes su longitud será manejable. Ahora, sin embargo, la máquina busca un path en el diccionario y el número de puntos elegidos está limitado por la longitud del string, mientras que el número de elecciones en cada punto está limitado por el número de lexicons en el diccionario. Dada una correspondencia léxico/superficie y un path en el lexicón, el traductor y el diccionario pueden verificar rápidamente si el par de strings léxico/superficie satisface todas las restricciones relevantes.

3.3.3. El efecto de los ceros. PSPACE completitud con cantidad de ceros irrestricta

Si no se restringe el uso de caracteres nulos, los argumentos de la sección anterior no son válidos. El problema es que el uso irrestricto de los caracteres nulos que modelan deleciones sin restricciones permiten que los strings de superficie y léxicos difieran fuertemente en su longitud. El tiempo que requiere buscar o verificar la correspondencia léxico/superficie puede no estar más acotado polinómicamente por el largo del string del input.

De hecho, el análisis en KIMMO con una cantidad irrestricta de ceros es probablemente PSPACE-hard (como mínimo tan complejo como cuaquier problema que puede resolverse en tiempo polinomial). También se muestra un sketch de un argumento para mostrar que pertenece a PSPACE, lo que es otra parte del argumento PSPACE-completitud. Luego en el peor caso el análisis en KIMMO resulta extremedamente difícil si no se restringe la cantidad de ceros.

La reducción mas fácil de PSPACE-completitud para el análisis del KIMMO con ceros irrestrictos involucra al problema computacional de Intersección de autómatas de estados finitos (Garey y Johnson [21]:226).

Una posible instancia del FSAI es un conjunto de autómatas finitos determinísticos sobre el mismo alfabeto. El problema es determinar cuando existe algún string que puede ser aceptado por todos los autómatas.

Dado un conjunto de autómatas sobre un alfabeto Σ, se construye el problema de análisis en KIMMO como sigue.

Sean a y b dos caracteres nuevos, no pertenecientes a Σ. Se define el nuevo alfabeto $\Sigma \cup \{a, b\}$.

Declaremos "@"como el caracter comodín y "0" como el caracter nulo.

Ahora se construye el resto del traductor en dos partes.

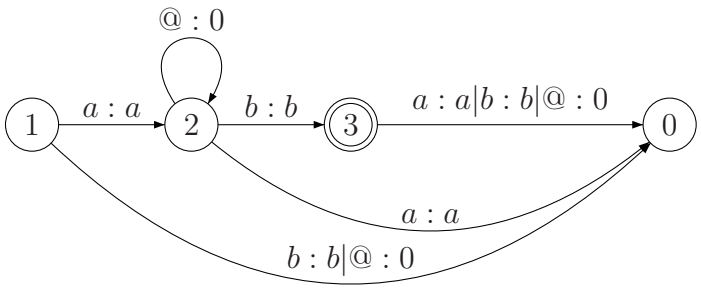
Primer paso

Se constuye el traductor driver principal:

```
"Main Driver"' 3 3

    a  b  @
    a  b  0

 1. 2  0  0   requiere a
 2. 0  3  2   deja correr el traductor
 3: 0  0  0   toma ab
```



Este traductor aceptará el string de superficie ab, permitiendo una cantidad arbitraria de giros con ceros antre a y b.

Segundo Paso

Cada traductor en el problema de FSAI, se translada directamente a un traductor de $KIMMO$:

1. Se aparean los caracteres originales de Σ con los caracteres nulos de la superficie.

2. También se agregan columnas para a/a y b/b con entradas cero a menos que se especifique.

3. Se elevan todos los números de los estados por dos.

4. Hagamos que el nuevo estado inicial acepte solamente a/a, pasando a 3 (el estado inicial del automata original)

5. Se define como estado final sólo al nuevo estado 2, pero para todo estado que era final en el traductor original, se hace una transición de 2 a b/b

Ejemplo 3.2

Por ejemplo si el autómata original tiene diagrama de estados:

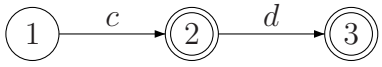


Figura 3.5: Autómata original

El traductor KIMMO obtenido será (se muestran las transiciones que llevan a estados finales):

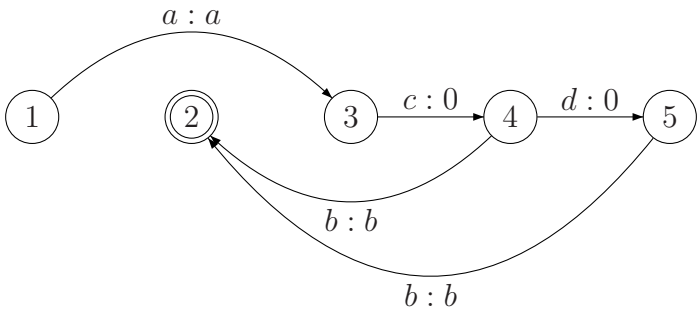


Figura 3.6: Traductor KIMMO

Tercer paso

Se hace que el léxico de raíces del diccionario contenga una entrada léxica para cada caracter individual en $\Sigma \cup \{a, b\}$. La clase de continuación de cada entrada debe enviar a esta atrás al léxico de raíces, exceptuando cuando la entrada de b debe listar la clase de continuación para #. Finalmente, se toma ab como el string de superficie para el problema de parsing en KIMMO.

Una a de superficie se levantará las versiones transladadas de los autómatas, los cuales podrán correr libremente entre a y b porque todos los caracteres en Σ están apareados con caracteres nulos en la superficie. Si existe algún string que sea aceptado por todos los autómatas originales, entonces este string léxico mandará todos

los autómatas traducidos en un estado donde la b restante sea aceptada. Por otra parte, si la intersección original es vacía, la b nunca será aceptada y el analizador no aceptará el string ab.

Como ejemplo, consideremos porque el analizador no siempre acepta el string ab por el recorrido del camino directo $1, 2$ y 3. El traductor driver aceptará de acuerdo a la secuencia, pero el par a/a manda cada autómata traducido al estado 3, lo que corresponde a su estado inicial antiguo. Por construcción, el estado 3 del autómata traducido no aceptará b/b salvo que el estado inicial fuera final en el autómata original. El analizador no puede invertir a par léxico/superficie salvo que todo autómata lo haga, luego no podemos aceptar ab directamente salvo que el estado inicial de todo autómata original fuera un estado final. Es decir, en el caso de que el string vacío es un string tal que todos los autómatas originales acepten y por tanto el analizador se comporta según lo deseado.

La contrucción anterior muestra que el análisis en KIMMO con una cantidad irrestricta de caracteres nulos es como mínimo tan compleja como cualquier problema en PSPACE pero para probar la completitud es también necesario probar que no es más complejo que los problemas en este conjunto.

Para esto, es suficiente con mostrar que se puede usar espacio polinomial mientras se transforman problemas de análisis en KIMMO en problemas FSAI.

Se considerará un problema de análisis arbitrario.

Primer paso

Se convierte el componente diccionario en un gran traductor tal que:

1. Restrinja los strings léxicos de la misma forma que lo hace el componente diccionario, apareando caracteres léxicos con caracteres comodín de superficie.

2. A pesar de esto, permita que los caracteres nulos sean insertados libremente en el nivel léxico, en el caso de que otros autómatas permitan nulos léxicos.

Esta conversión puede ser realizada pues el componente diccionario es de estados finitos.

Segundo paso

Se convierte también el string input en un autómata. El autómata del string-input debe:

1. Restringir el string de superficie para que sea exacatamente el string input.

2. Permitir que los nulos de superficie sean insertados libremente.

Tercer paso

Se expanden todos los comodínes y los caracteres de subsets en el autómata, luego se interpreta cada par léxico/superficie en la cabeza de una columna de un autómata como un caracter individual en un alfabeto extendido. Dado este procedimiento, es posible resolver el problema original de análisis para resolver el FSAI para el conjunto aumentado de autómatas. Como el string input es ahora codificado por un autómata la intersección de todos los lenguajes aceptados por todos los autómatas consistirá de todas las correspondencias léxico/superficie permitidas que reflejan el análisis del string input. La intersección sera no vacía si y sólo si el sting input es reconocible.

3.4. La Respuesta de Church y Koskenniemmi

La gramática artificial desarrollada por Barton [5] para demostrar que el problema de generación en KIMMO es equivalente al problema de SAT puede pensarse como un set de procesos de armonía de tantos rasgos como variables se consideren en la fórmula. Que para cada armonía el valor del rasgo sea respectado en toda la fórmula está asegurado por cada traductor de consistencia. Koskenniemi y Church [42] tomán este punto de partida para criticar el análisis de Barton como no pertinente al procesamiento de lenguajes naturales. Estos autores destacan que el resultado de complejidad de Barton es no estándard porque describe la complejidad de problema en función del tamaño de la gramática más que el costo de espacio-tiempo en función del tamaño del input. Este último enfoque es el usual porque usualmente el peso del tamaño de la gramática es pequeño porque el número de reglas es fijo. Este sería el caso de las lenguas naturales donde las gramáticas son de pequeño tamaño. La codificiación de Barton no tiene análogos en las lenguas naturales porque estas tienen a los sumo dos procesos de armonía. Asimismo la complejidad del análisis con la codificación de Barton es lineal en el tamaño del input cuando la fórmula es de una sola variable que es lo que normalmente ocurre con los procesos de armonía en las lenguas del mundo.

Como ejemplo de lo que ocurre con un gramática de una lengua natural consideran el caso del finés. Ofrecen datos experimentales correspondientes al análisis de un corpus obtenido de dos muestras provenientes de un diario finlandés. Grafican

el tiempo de reconocimiento (cantidad de pasos) contra el largo de palabras (en cantidad de letras) y realizan una regresión lineal obteniendo una pendiente de 2.43, según su punto de vista con buen ajuste.

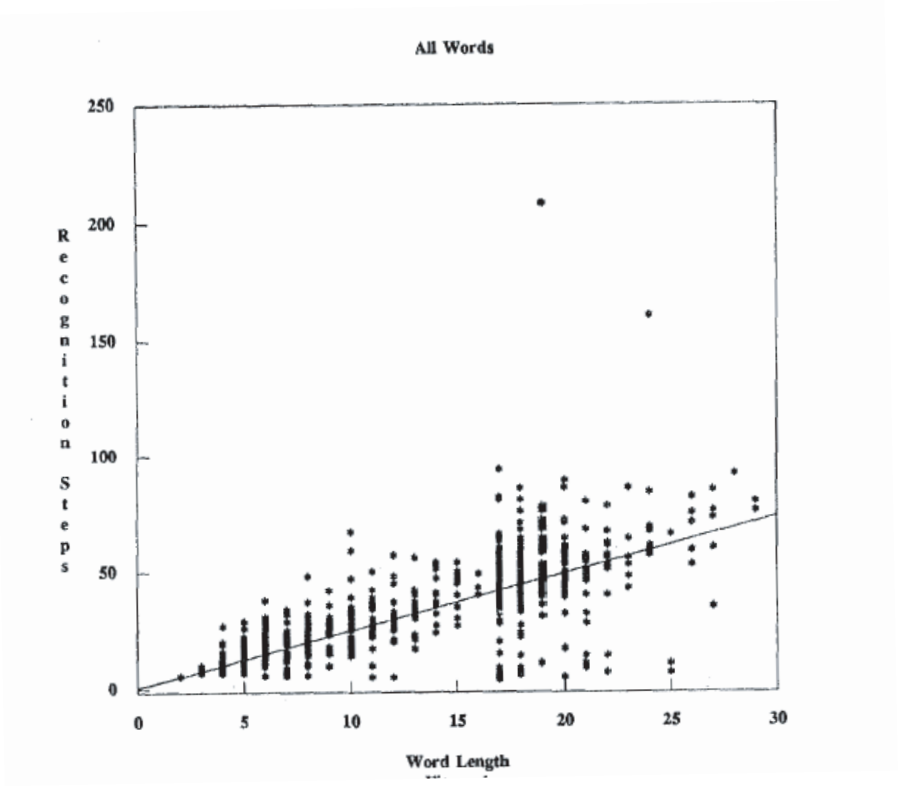


Figura 3.7: Longitud de palabra vs. tiempo de reconocimiento (de Church y Koskenniemi [42]))

Capítulo 4

Autómatas finitos de dos cabezas y lenguajes lineales. Su posible aplicación al modelado de lengua con algunos procesos morfológicos complejos

4.1. Introducción

En nuestra búsqueda de un modelo que nos permitiera expresar naturalmente la morfología verbal de la lengua toba (guaycurú) encontramos algunos escollos que surgen de ciertas características de esta lengua, situación que parece ser muy común en muchas lenguas americanas. Existen muchos lenguajes naturales que tienen morfologías que hacen uso conjunto de la sufijación y de la prefijación y para los cuales existen interdependencias entre los dos tipos de afijos. Se muestra en este capítulo la insuficiencia del modelo concatenativo usual para modelar estos procesos morfofonológicos. Se presenta también la propuesta de Creider, Hankamerz y Wood [19] para el modelado de estos fenómenos en términos de lenguajes lineales libres de contexto y traductores de dos cabezales.

4.2. Algunas lenguas para las cuales no parece ser apropiado un modelo por autómatas finitos de una cinta

A pesar de que gran cantidad de procesos morfológicos pueden ser modelados por autómatas finitos, existen gran cantidad de procesos que no pueden ser manejados de manera óptima mediante estos dispositivos. Consideremos el adjetivo derivado enlatable. Para obtener la derivación del mismo a partir del nombre lata se debe suponer la existencia de dos ítems léxicos diferentes obtenidos al marcar lata para cada una de las instancias. El autómata que modela esto sería:

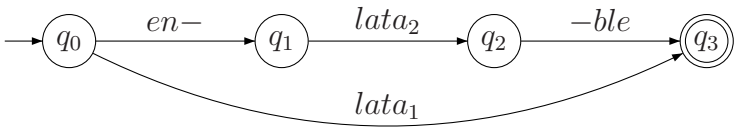


Figura 4.1:

El problema de la independencia entre prefijo y subfijo es muy común. Una gran cantidad de lenguajes presentan morfosintaxis que utiliza prefijos y subjfijos con fenómenos de interdependencia entre los mismos.

El toba es una lengua de la familia Guaycurú, hablada en el Chaco argentino y paraguayo, con una cantidad aproximada de entre 30.000 y 50.000 hablantes (Messineo [44]). Presenta tres set de marcadores de personas. El marcador empleado es en ciertas ocaciones determinado por el grado de afectación por la acción de la persona que la realiza. En el siguiente ejemplo la aglutinación del sufijo de reflexivo (-l'at) implica la marcación de la persona activa con subifjos del la clase correspondiente a la voz "media"porque el agente es afectado por la acción.

Ejemplo 4.1 .

(a)	*y-*	*alawat*	
	3Activa	*-matar*	
	¨él	*mata"*	
(b)	*n-*	*alawat*	*-l'at*
	3Media-	*matar*	*-reflexivo*
	¨él se	*suicida"*	

En los procesos de derivación de nombres a partir de verbos se utilizan subfijos. Los nombres llevan prefijos que son marcadores persona poseedora de nombres.

Ejemplo 4.2 .

l-	*edaGan*	*-at*
3pos	*escribir*	*instr*
	"lápiz(de él)"	

Ejemplo 4.3 .

ne-	*peGagenataGan*	*-aGa*
4pos-	*enseñar*	*-ag*
"la maestra (de alguien)"		

En relación al ejemplo anterior debe observarse que en el toba una clase muy grande de sustantivos debe llevar marca de poseedor obligatoria.

En el maká, lengua de la familia lingüística mataguaya, hablada en el Paraguay, los verbos derivados de otros por medio del subfijo transitivizador-causativizador -inen, se exige la marcación de persona con el prefijo en caso ergativo (Gerzenstein[22]):

Ejemplo 4.4 .

(1)	*he-*	*wel*
	Suj1	*subir*
	"Yo	*subo(intr.)"*

Ejemplo 4.5 .

(2)	*hi-*	*wel*	*-inen*	*n*	*-a'*	*y-*	*as*
	A1-	*subir*	*caus*	*dem*	*-masc*	*pos1*	*hijo*

"Yo hago subir a mi hijo"

Tanto en la aplicación de desiderativos-causativos como en la de desiderativos-causativos, la aplicación de los subfijos respectivos exige la prefijación del -Vq- (V, vocal determinada por reglas de armonía):

Ejemplo 4.6 .

(1)	*y-*	*akha'*	*ho-*	*n-*	*okon*	*-le*	*yi-*	*tem*
	1-	*Pers*	*Suj1*	*r*	*rascar*	*r*	*p1*	*r*

"Yo me rasco (a mí mismo)"

Ejemplo 4.7 .

(2)	*ts-*	*oq-*	*okon*	***-heyu***	*-le*	*yi-*	*tix*
	Suj1	*pr*	*rascar*	***des***	*r*	*p1*	*r*

"Tengo ganas de rascarme (a mí mismo)"

En el Wichí o Mataco, lengua de la misma familia, hablada en Argentina y en Bolivia por una cantidad de hablantes estimada de entre 35.000 y 65.000 hablantes Censabella[15] utiliza sufijos y prefijos en su morfología. Según Viñas de Urquiza [54] el sufijo -hat, causativo, se utiliza para derivar verbos a partir de nombres. Estos verbos derivados marcan la persona que realiza la acción mediante prefijos verbales.

Ejemplo 4.8 .

pelah: blanco.

o-*pel****-hat****:*** *yo blanqueo.*

Esta lengua presenta un conjunto de sustantivos dependientes. Esta clase esta caracterizada por el hecho de que todo elemento lleva una marca obligatoria de poseedor que es un prefijo. En los ejemplos siguientes se obtiene a partir de un sufijo nominalizador un sustantivo de esta clase Hunt[31]:

Ejemplos 4.9 .

humin: amar.

O-*humna****-yaj****:*** *mi amor.*

chote: ayudar.

O*-chot-****yaj****:*** *mi ayuda.*

En el Kamaiurá, lengua de la familia Tupí-Guaraní, hablada en el Mato Grosso, con una cantidad aproximada de 300 hablantes, el prefijo marcador de persona del verbo pertenece a un set de prefijos que se predice a partir del subfijo de modo (Seki [50]):

Ejemplo 4.10 .

Modo	*Afijo forma positiva*	*Afijo forma negativa*	*Marcador de persona*
Indicativo	∅	*na=...-ite*	*Set I, II*
Imperativo	∅	*-em*	*Serie III*
Exortativo	*ta=...=in*	*-um*	*=Indicativo*

El guaraní yopará, lengua oficial de la República del Paraguay junto con el castellano, pertenece a la misma familia que la lengua anterior. Tenía, en 1995, una cantidad estimada de 4.648.000 hablantes (Censabella [15]). Tiene una rica morfología que utiliza tanto prefijos como sufijos y presenta además fenómenos de armonía nasal que desde la raíz se expanden bidireccionalmente hacia los afijos. Esta lengua utiliza prefijos verbales para marcar la persona agente y tiene algunos verbos derivados a partir de nombres utilizando el sufijo -a (Krivoschein[43]y Guasch [24]) .

Ejemplo 4.11 .

akã : cabeza.

a *− akã −* ***'o****: decapito.*

o *− akã −* ***'o****: él decapita.*

En un trabajo que trata sobre el modelado de estos procesos, Creidery, Hankamerz y Woodx [19], presentan un ejemplo clave. El nandi, lengua de la familia, presenta un fenómeno de armonía vocálica con respecto al rasgo ATR (Advance tongue root) que da el grado de adelantamiento de la vocal en su emisión. Un gran número de raíces y subfijos deben ser especificados en el lexicón con el valor [+ATR]. Estos ítems afectan todos los prefijos y subfijos subsecuentes, haciendo que sus vocales sean articuladas con el rasgo [+ATR]. Como los prefijos determinan el valor del rasgo del prefijo y nunca al revés, se pueden obtener ejemplos que muestran que los prefijos son parseados antes que los prefijos.

Ejemplo 4.12 .

(a)	*kI:-*	*gO-*	*sO:man*
	Pas	*3P*	*leer*
	él o	*ellos*	*han estudiado (tiene educación)*

Ejemplo 4.13 .

(b)	*ki:-*	*go-*	*so:man*	*-i*
	Pas	*3P*	*leer-ASP*	*Impf*
	él o	*ellos*	*habían estado*	
			estudiando	

En (b) la presencia del subfijo de aspecto imperfectivo especificado con el valor [+ATR] induce que todas las otras vocales cambien a sus equivalentes con el rasgo [+ATR] y por tanto el subfijo debe ser parseado antes que los prefijos.

La propuesta de Creider, Hankamerz y Wood [19] consiste en modelar la morfosintaxis de este tipo de lenguas por medio de autómatas no determinísticos de dos cabezales que son los aceptores de los lenguajes lineales libres de contexto. Así se liberan del requerimiento de lectura de izquierda a derecha. Si suponemos identificada la raíz se tiene un string compuesto por los prefijos (tomados como símbolos atómicos) la raíz y los subfijos:

$$p_n + \ldots + p_1 + [\text{raíz}] + s_1 + \ldots + s_n$$

Separando las dos cadenas se obtienen:

$p_n + \ldots + p_1$

y

$s_1 + \ldots + s_n$

Se hace conveniente para modelar las interdependencias obtenidas entre los substrings de arriba trabajar con un autómata con dos cabezales. Este autómata lee una cinta que contiene los prefijos y otra que contiene los sufijos (ver figura 4.2). Las lecturas se hacen independientemente, en cada transición se aplica solamente una cinta. Si no hay transición en una cinta, la cabeza de lectura asociada no se mueve. Cuando ambas cintas has sido procesadas, si el autómata está en un estado final, el string se acepta.

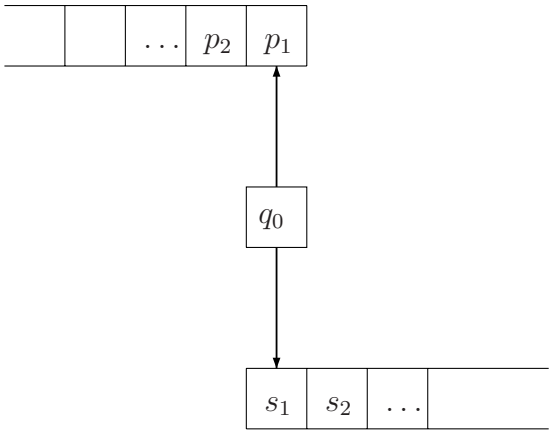


Figura 4.2: Posición inicial de un autómata de dos cintas

Los autómatas de dos cintas fueron investigados por Rosenberg [46] quien probó la equivalencia entre los lenguajes aceptados por estos dispositivos y los descriptos por las gramáticas libres de contexto lineales. Estas gramáticas son, según Rosenberg: "La clase más baja de lenguajes libres de contexto no regulares". Así, según los autores indican, estas gramáticas se acomodan a la descripción requerida sin apartarse radicalmente del modelo de morfología de dos niveles. En la figura 4.3 se observa como se puede modelar la derivación del adjetivo *enlatable* a partir del nombre *lata*.

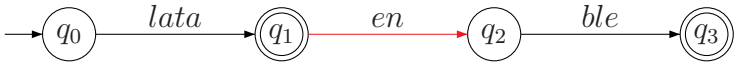


Figura 4.3: Modelado de la derivación de *enlatable* a partir de *lata* con autómatas de dos cintas. Las transición en rojo marca la del cabezal que lee prefijos.

4.3. Lenguajes lineales libres de contexto

Definición 4.14 *Un lenguaje lineal libre de contexto es un lenguaje generado por una gramática* $G = (N, \Sigma, P, S)$ *de tipo 2 o libre de contexto donde toda regla* P *es un elemento del conjunto* $N \times (N \cup N\Sigma \cup \Sigma N)$ *y donde* N *es el conjunto de los no terminales,* S *es un símbolo para sentencias.*

Ejemplo 4.15 .

La gramática $G = (N, \Sigma, P, S)$

$$\begin{aligned} N &= \{S, A\} \\ \Sigma &= \{a, b\} \\ P: \quad & S \to aA \\ & A \to Sb \\ & S \to \lambda \end{aligned}$$

es una gramática lineal libre de contexto.

Observación 4.16 .

$\Delta = L(G) = \{a^n b^n | n \geq 0\}$

Proposición 4.17 *La clase de lenguajes lineales (LL), contiene propiamente a la de los lenguajes regulares (LR) y, además, está contenida propiamente en la de los lenguajes libres de contexto(LLC).*

Demostración: .

LR $\subset$ LL)

La contención está dada por la definición de lenguaje lineal, que tal contención es propia se obtiene, por ejemplo, considerando el ejemplo anterior.

LL $\subset$ LLC)

También surge de la definición de LL. Para probar que es propia consideraremos al lenguaje $\Delta\Delta$.

$\Delta\Delta \in LLC$, en efecto un conjunto posible de reglas que lo generan puede ser:

$$\begin{aligned} N &= \{X_1, X_2, A_1, A_2\} \\ \Sigma &= \{a, b\} \\ P: \quad & S \to X_1 X_2 \\ & X_1 \to aA_1 \\ & A_1 \to X_1 b \\ & X_1 \to \lambda \\ & X_2 \to aA_2 \\ & A_2 \to X_2 b \\ & X_2 \to \lambda \end{aligned}$$

Por otra parte, $\Delta\Delta \notin LR$, para probar esto usaremos el siguiente lema:

Lema 4.18 *Lema de Pumping para lenguajes lineales. Sea L un lenguaje lineal, entonces exite un $n_0 \in N$ tal que si $s \in L$ y $|s| \geq n_0$ entonces existen v, w, x, y, z tales que:*

1. $s = vwxyz$
2. $|vwyz| \leq n_0$
3. $vw^n xy^n z \in L, \forall n \geq 0$
4. $wy \neq \lambda$

Consideremos ahora $s = a^{n_0} b^{n_0} a^{n_0} b^{n_0}$, entonces por 2. se tiene: $v = a^{A_1}$ $w = a^A$ $y = b^B$ $z = b^{B_1}$

□

Reemplazando en la expresión de s, se tiene:

$$s = a^{A_1} a^A x b^B b^{B_1} \Rightarrow x = a^{A_2} b^{n_0} a^{n_0} b^{B_2}$$

Ahora, por 3), se tiene:

$$s_M = a^{A_1} (a^A)^M a^{A_2} b^{n_0} a^{n_0} B b^{B_2} (b^B)^M b^{B_1} \in L, \forall M \geq 0$$

con A o $B \neq 0$. Sin restricción de generalidad, podemos suponer, $A \neq 0$ y tomando M suficientemene grande se tiene que:

$$a^{A_1} (a^A)^M a^{A_2} b^{n_0} \notin \Delta$$

y por tanto:

$$s_M \notin \Delta\Delta$$

4.4. Los autómatas finitos de dos cabezas como aceptores de lenguajes lineales libres de contexto

Definición 4.19 *Un autómata finito de dos cabezas M esta definido por una tupla $(Q, \Sigma, \delta, s, F)$, donde:*

Q es un alfabeto de estados.

Σ es un alfabeto para el input y $\Sigma^\lambda = \Sigma \cup \{\lambda\}$.

$\delta \subseteq Q \times \Sigma^\lambda \times \Sigma^\lambda \times Q$ es una relación finita de transición.

$s \in Q$ es un estado inicial.

$F \subset Q$ es un conjunto de estados finales.

Se interpreta una transición de la forma (p, a, λ, q) indicando que si el estado actual del autómata M es p y el símbolo en el cabezal izquierdo de lectura es a luego M va a estado q y mueve el cabezal izquierdo un símbolo a la izquierda. Similarmente una transición de la forma (p, λ, a, q) mueve la cabeza derecha a la derecha, haciendo que el autómata M pase del estado p a q. Se permiten transiciones que mueven ambos cabezales sólo si el estado en vigencia es el inicial. Para inicializar M movemos los dos cabezales de lectura para que estén sobre el mismo símbolo en el string input (si el string input es λ, el autómata termina inmediatamente). La elección en la aparición de los símbolos es hecha de manera no determinística. Luego, una transición del estado inicial tiene la forma (s, a, a, p) donde p es un estado a analizar y M mueve ambos cabezales, el de la derecha hacia ese mismo lado y el de la izquierda, en esa misma dirección.

Para definir secuencias computables en M, definimos primero sus configuraciones.

Definición 4.20 *Una tupla $(p, x, a, b, y) \in Q \times \Sigma^* \times \Sigma^\lambda \times \Sigma^\lambda \times \Sigma^*$ es una configuración de M.*

Esto se interpreta como que el cabezal izquierdo está sobre a, el cabezal derecho sobre b, xa es un prefijo no leído del string input y by un sufijo no leído del input. Si $p = s$ luego $a = b$ y xay es el string input, de otra manera a o b son el string vacío. Una configuración (s, x, a, a, y) se llamará un configuración inicial y una configuración $(f, \lambda, \lambda, \lambda, \lambda)$ se denominará configuración final.

M tiene tres tipos de pasos computacionales:

1. M es una configuración inicial (s, x, a, a, y) y $(s, a, a, P) \in \delta$ luego, una nueva configuración nueva de M es (p, x', c, d, y'), donde $x = x'c$ y $y = dy'$. Se escribe $(s, x, a, a, y) \vdash (p, x', c, d, y')$.

2. M es una configuración (p, x, a, b, y) con $p \neq s$ y $(p, a, \lambda, q) \in \delta$ luego, una nueva configuración de M es (q, x', c, b, y), donde $x = x'c$. Se escribe $(p, x, a, b, y) \vdash (q, x, c, b, y')$.

3. M es una configuración (p, x, a, b, y) con $p \neq s$ y $(p, \lambda, b, q) \in \delta$ luego, una nueva configuración de M es (q, x, a, d, y'), donde $y = dy'$. Se escribe $(p, x, a, b, y) \vdash (q, x, c, d, y')$.

Se extiende $\vdash$ a $\vdash^i, i \geq 0$, $\vdash^+$ y $\vdash^*$ en el sentido usual.

Un string $w \in \Sigma^*, w = xay$ es aceptado por autómata de dos cabezales si existe una computación:

$$(s, x, a, a, y) \vdash^* (f, \lambda, \lambda, \lambda, \lambda),$$

con $f \in F$. Notamos, como siempre, por $L(M)$ al lenguaje aceptado por M.

Considermos una gramática lineal libre de contexto es una upla $G = (N, \Sigma, P, S)$, donde:

Escribiremos $uAV \Rightarrow u'Bv'$ si tanto $A \rightarrow aB \in P$, $u' = ua$ y $v' = v$ o $A \Rightarrow Bb \in P$, $u' = u$y $v' = bv$. En adición, se escribe $uAv \Rightarrow uav$ si $A \Rightarrow a \in P$.

Se extiende $\Rightarrow$ a $\Rightarrow^i$, $i \geq 0$. $\Rightarrow^+$ y $\Rightarrow^*$. Un string terminal x es una sentencia de G si $s \Rightarrow^* x \in G$. El conjunto de sentencias del lenguaje se nota como $L(G)$.

Observese que tanto la definición de lenguaje lineal como de autómata no contienen al string vacío. Esta restricción, aunque conveniente desde el punto de vista técnico, no es necesaria.

Teorema 4.21 *Todo lenguaje lineal es el lenguaje aceptado por un autómata de dos cabezales.*

Demostración: Sea $L \subseteq \Sigma^*$, donde Σ es algún alfabeto, un lenguaje lineal. Luego $L = L(G)$ para alguna gramática lineal $G = (N, \Sigma, P, S)$.

Se construye el autómata $M = (N \cup \{s\}, \Sigma, \delta, s, \{S\})$ tal que $L(M) = L$. La construcción es la siguiente:

1. (s, a, a, A) es una transición de inicio si $A \to a$ es una producción en G.

2. (B, a, λ, A) es una transición de continuación si $A \to aB$ está en G.

3. (B, λ, a, A) es una transición de continuación si $A \to aB$ está en G.

Se afirma que $(s, x, a, a, y) \vdash^i (s, \lambda, \lambda, \lambda, \lambda)$ si y sólo si $S \Rightarrow^i xay$. Se prueba esta afirmación por inducción en el número de pasos y por extensión directa del resultado. El resultado extendido es: $(s, x, a, a, y) \vdash^i (A, \lambda, \lambda, \lambda, \lambda)$ si y sólo si $A \Rightarrow^i xay$.

El caso base para $i = 1$ sigue directamente de la construcción, es decir,$(s, x, a, a, y) \vdash (A, \lambda, \lambda, \lambda, \lambda)$ si y sólo si $A \Rightarrow a$, es decir si $A \to a$ está en P.

Se supone ahora que el argumento vale para algún $i \geq 1$ y se demostrará que vale para computaciones y derivaciones de longitud $i + 1$. Si $(s, x, a, a, y) \vdash^{i+1} (A, \lambda, \lambda, \lambda, \lambda)$, luego, como $i+1 \geq 2$, o bien $(s, x, a, a, y) \vdash^i (B, \lambda, \lambda, b, \lambda)$ o $(s, x, a, a, y) \vdash^i (B, \lambda, \lambda, \lambda, \lambda)$ para algun estado a y símbolo b. Ahora $(s, x, a, a, y') \vdash^i (B, \lambda, \lambda, \lambda, \lambda)$ donde $y = y'a$, luego, $B \Rightarrow^i xay'$, por induccion en la hipótesis. Como $(s, x, a, a, y') \vdash^i (B, \lambda, \lambda, \lambda, \lambda)$,$A \to Bb$ y $B \Rightarrow^{i+1} xay'$, como se quiere demostrar.

Inversamente, si $A \Rightarrow^{i+1} xay'$, luego como $i + 1 \geq 2$, $A \Rightarrow Bb \Rightarrow^i xay$ para algun b y B. Nuevamente solo se considerará la primera posibilidad. Claramente $B \Rightarrow^i xay'$ donde $y = y'b$ y $(s, x, a, a, y') \vdash^i (B, \lambda, \lambda, \lambda, \lambda)$. Luego $(s, x, a, a, y) \vdash^{i+1} (A, \lambda, \lambda, \lambda, \lambda)$.

□

Teorema 4.22 *Todo lenguaje aceptado por un autómata de dos cabezales es un lenguaje lineal* $M = (Q, \Sigma, \delta, s, F)$.

Demostración: Sea $L \subseteq \Sigma^*$ un lenguaje aceptado por un autómata M, con Σ el alfabeto. Luego $L = L(M)$ para algún autómata $M = (Q, \Sigma, \delta, s, F)$. Se contruye una gramatical lineal $G = (Q \cup \{S\}, \Sigma, P, S,)$ tal que $L = L(G)$ definida por:

1. $p \to a$ es una producción terminal si (s, a, a, p) es una transición en M.

2. $q \to ap$ es una producción si (p, a, λ, q) es una transición en M.

3. $q \to pa$ es una producción si (p, λ, a, q) es una transición en M.

4. $S \to ap$ es una producción si (p, a, λ, q) es una transición en M para algún estado final q.

5. $S \to pa$ es una producción si (p, λ, a, q) es una transición en M para algún estado final q.

Se afirma que $S \Rightarrow^i xay$ si y sólo si $(s, x, a, a, y) \vdash^i (A, \lambda, \lambda, \lambda, \lambda)$ para algun $p \in F$. Se establece primero, por inducción en el número de pasos, que $p \Rightarrow^i xay$ si y sólo si $(s, x, a, a, y) \vdash^i (p, \lambda, \lambda, \lambda, \lambda)$. Luego si $p \in F$, por la construcción de G, se puede reemplazar el primer paso que usa una $p-$producción con una $S-$producción que tiene el mismo lado derecho y la aseveración vale. El caso base de la inducción para $i = 1$. Luego $(s, \lambda, a, a, \lambda) \vdash (p, \lambda, \lambda, \lambda, \lambda)$ en M y, por construcción $p \Rightarrow a$ en G cuando p está en F. Luego $S \Rightarrow a$.

Consideremos una computación $(s, x, a, a, y) \vdash^{i+1} (p, \lambda, \lambda, \lambda, \lambda)$. Luego el paso i+1-ésimo es $(q, \lambda, b, \lambda, \lambda) \vdash (p, \lambda, \lambda, \lambda, \lambda)$ o bien $(q, \lambda, \lambda, b, \lambda) \vdash (p, \lambda, \lambda, \lambda, \lambda)$ para algún $q \in Q$. Sin restricción de generalidad se considera solamente el primer caso:

Claramente $(s, x', a, a, y) \vdash^i (q, \lambda, \lambda, \lambda, \lambda)$, donde $x = bx'$ y $q \Rightarrow^i x'ay$ en G.

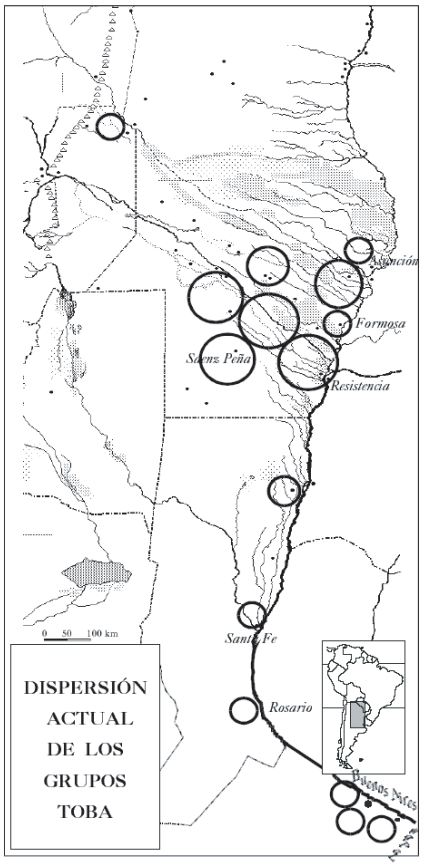
Es más, $p \to bq$ está en P y , por lo tanto, $p \Rightarrow^{i+1} xay$ y con lo que se completada el paso inductivo y toda la demostración pues $S \Rightarrow^i xay$ si p está en F.

□

4.5. Descripción de la morfología verbal del toba usando autómatas finitos de dos cabezas

La lengua toba pertenece, junto con el pilagá, el mocoví y el caduveo a la familia lingüística guaycurú (Messineo[44]). A esta familia pertenecen también las ya extintas lenguas abipón y mbayá.

En la actualidad, el toba se habla en la región del Gran Chaco (Argentina, Bolivia y Paraguay) y en asentamientos permanentes denominados barrios en las ciudades de Resistencia, Presidencia Roque Sáenz Peña, Rosario, Santa Fe, Buenos Aires y La Plata (Argentina).



Mapa 2. Los círculos señalan áreas donde se encuentran las distintas localidades, no indican cantidad de hablantes.

Figura 4.4: Distribución actual de los grupos toba, según Messineo[44]

Desde el punto de vista sociolingüístico, puede considerarse una lengua en peligro ya que en ciertos ámbitos comunicativos ha sido desplazada por el español y su transmisión intergeneracional -especialmente en las comunidades urbanas- está interrumpiéndose. El alfabeto en uso en la lengua es el propuesto por el pastor Alberto Buckwalter. Este misionero menonita es autor de importantes trabajos lingüísticos, entre ellos una traducción del nuevo testamento a las lenguas toba y pilagá, un Vocabulario toba [13] y otro Mocoví [12]. El alfabeto en uso es, sin embargo, no fonológico y utiliza, además, muchos digramas. Teniendo en cuenta este inconveniente preferimos utilizar la signografía utilizada por Messineo [44] en su descripción. En las siguientes cuadros se muestran las consonantes y vocales de la lengua:

CONSONANTES	labiales		alveolares		palatales		velares		uvulares		laríngeas	
	sor	son	sor	son	sor	son	sor	son	sor	son	sor	son
oclusivas	p		t	d	c		k	g	q	ɢ	ʔ	
fricativas			s		ʃ	ʒ					(h)	
nasales		m		n		ɲ						
laterales				l		ʎ						
semiconsonantes		w				y						

Figura 4.5: Cuadro fonológico de consonantes (Messineo [44])

Vocales	
i	o
e	
a	

Figura 4.6: Cuadro fonológico de vocales (Messineo [44])

El toba, es una lengua que desde el punto de vista de su tipología morfológica presenta caracterisiticas de lengua aglutinante polisintética. Messineo [44] cita a

tal respecto que en el verbo es posible codificar participantes y relaciones gramaticales, de tal manera que una sola palabra es posible expresar una oración completa. Considera como ejemplo al verbo *sanadatema*:

Ejemplo 4.23 .

s-	*anat(a)*	*-d*	*-em*	*-a*
1A-	*aconsejar*	*2 dat*	*ben*	

Según esta autora el verbo consituye la clase de palabra morfológicamente más compleja. Las categorías gramaticales de persona se prefijan al tema verbal y señalan persona, persona/número y caso semántico. Se utilizan sufijos para marcar el plural de algunas personas como así también otras categorías gramáticales como el aspecto, la dirección-locación de la acción, la marca de reflexivo y de recíproco y el modo desiderativo. El verbo no tiene marca de tiempo.

Una de los rasgos característicos del toba es el sistema de marcación Activo-Inactivo en los prefijos verbales (Messineo[44] y Klein [40]). Existen en esta lengua dos sets de prefijos verbales que marcan acción:

1. Tipo I (In): codifica participantes inactivos, objeto de verbos transitivos y pacientes de verbos intransitivos.

2. Tipo II (Ac): señala participantes activos, sujeto de verbos transitivos e intransitivos.

   Activo afectado (M): codifica la presencia de un participante activo afectado por la acción a la que hace referencia el verbo.

El número de verbos que llevan esta marca de tipo I están limitados a aproximadamente 20 pero serán considerados con el mismo peso que los otros porque podrían ser relevantes para nuestro objetivo de encontrar una gramática que describa algunos aspectos de esta lengua.

4.5.1. Análisis del paradigma verbal de primera y segunda persona

Consideraremos en primer lugar el caso de la primer y segunda persona por ser más sencillos. Para la primer y segunda personas los marcadores son:

	I (inactivo)	II Activo	II Activo afectado
1.SG	3(V)-	s(V)-	ñ(V)-
2.SG	?ad-	?aw-	?an-
1.PL	qad-	s(V)-	ñ(V)-
2.PL	qad-...-i	qaw-...-i	qan-...-i

A partir de esto contruímos los autómatas que modelan la marcación con cada uno de estos paradigmas.

Por otra parte teniendo en consideración que la aplicación del reflexivo obliga la marcación con prefijos del tipo II, Activo afectado, los procesos de derivación de verbo a nombre y viceversa (ver ejemplos), como así también la flexión de la tercera persona que será considerada más adelante consideramos que desde el punto de vista computacional es más natural una descripción de la morfología de esta lengua en términos de autómatas de dos cabezas.

1. Subfijos Causativos (Caus). Los subfijos de causativo dado por los sufijos y sus alomorfos se aglutinan inmediatamente después de la raíz verbal. Se han contabilizado hasta 4 aglutinaciones sucesivas de sufijos, estos no tienen influencia en la clase de prefijo marcador de persona utilizado en cuanto cuando el análisis se restringe a la primer y segunda personas.

   Ejemplo 4.24 .

   seque'e: yo como.
   sequi'aGan: yo le doy de comer (lit. le hago comer)

2. Subfijos Plurales

3. Asp: los sufijos que marcan aspecto ocurren antes que los que marcan lugar y dirección y después de las marcas de plural de agente y de los causativos. Estos sufijos pueden marcar aspecto puntual (-(V)n), aspecto durativo (-ta), aspecto progresivo (-tak y alomorfos), aspecto iterativo (-i?)

4. Dir: los sufijos direccionales marcan la dirección de la acción

   a) hacia el interior: $-wek$

   b) hacia afuera: $-wo$

   c) hacia arriba: $-šigem$

d) hacia abajo: $-\tilde{n}i$

e) hacia el agua: $-aGasom$

f) hacia el fuego:$-waq$

5. Loc: los sufijos locativos expresan la ubicación relativa de los referentes argumentales. Su box es contiguo y posterior al de los direccionales

   a) sobre: $-lek$

   b) en el interior de: $-gi$

   c) debajo (escondido) de $-?ot$

   d) debajo (a la vista)de: $-asop$

   e) en un lugar determinado: $-a$

   f) al lado: $-at$

   g) lugar a poca distancia: $-i?$

   h) orientado a: $-ge$

   i) enfrentado, opuesto: $-get$

6. Recip: sufijo que indica accción recíproca: $(V)a?at$, la aplicación de este sufijo implica el uso de prefijos de persona correspondiente al tipo de Activo afectado.

7. Refl: sufijo de acción reflexiva, es decir la acción del verbo cae sobre la persona que realiza la acción: $l?at$. Al igual que el sufijo anterior implica el uso de un marcador del tipo Activo Afectado.

8. Sufijo de negación: El prefijo de negación, sa-, ocupa el primer box en la palabra, contiguo y antecediendo al de la marca de persona.

   Teniendo en cuenta el orden y las mutuas restricciones entre los sufijos y prefijos construímos el autómata tentativo para describir la morfología verbal del toba para las primer y segunda persona. Obsérvese que la marca de persona se determina casi en última instancia (con el cabezal que lee prefijos) para considerar tener en cuenta la acción de los reflexivos y recíprocos.

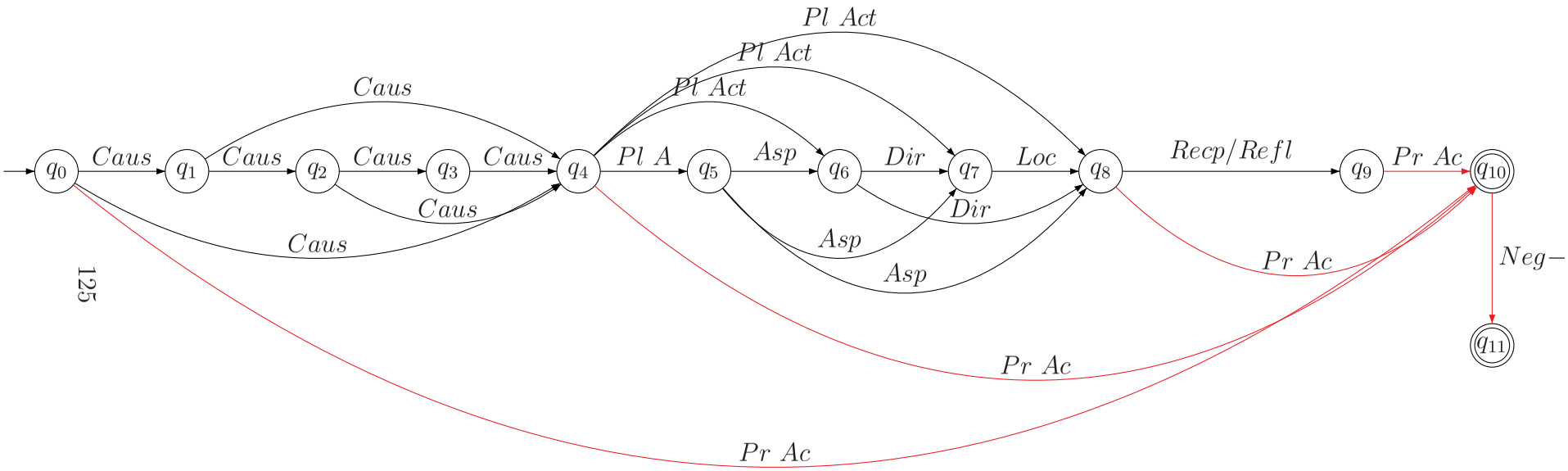


Figura 4.7: Esquema de la morfología verbal del la primer y segunda persona del toba.
Las líneas en negro y rojo marcan las transiciones del cabezal que leen sufijos y prefijos, respectivamente

4.5.2. Análisis del paradigma verbal de tercera persona

El caso de la marca para la tercera persona parece ser más complicado en el toba y en todas las lenguas de la familia guaycurú. Aquí la transitividad del verbo parece jugar un importante papel en la marca seleccionada. Los marcadores correspondientes son Messineo[44]:

a) i-/y- para verbos transitivos y sujetos de algunos intransitivos

b) d(V) para verbos tipicamente intransitivos.

c) n sujetos de verbos medios.

La sucesiva aplicación de los causativos parece actuar aquí, según la interpretación de Buckwalter, como haciendo switch en la transitividad del verbo original. Esto se puede observar en los ejemplos 4.25 y 4.26 (Buckwalter [13]:350):

Ejemplo 4.25 .

VI	*de-*	*que'e*		*come*
VT	*i-*	*qui'*	*-aGan*	*le da de comer*
VI	*de-*	*qui'*	*-aGanataGan*	*da de comer*
VT	*i-*	*qui'*	*-aGanataGanaGan*	*le manda dar de comer*
VI	*de*	*qui'*	*-aGanaGanataGan*	*manda dar de comer*

Ejemplo 4.26 .

VI	*do-*	*'ochi*		*duerme*
VT	*i-*	*'ochi*	*-aqchit*	*le hace dormir*
VI	*do-*	*'ochi*	*-aqtaxan*	*hace dormir*
VT	*i-*	*'och*	*-aqtaxanaxan*	*le manda hacer dormir*
VI	*do*	*'ochi*	*-aqtaxanataxan*	*manda hacer dormir*

Para modelar este comportamiento con nuestros autómatas de dos cabezales definimos dos recorridos determinados por la transitividad del verbo resultante después de la adjunción de los causativos. Estos recorridos quedan definidos por la paridad de la cantidad de causativos aglutinados. A partir de la tercera persona también se forma la tercera persona de actor indefinido a partir de un

prefijo, *qa-*, que es anterior y contiguo a la marca de tercera persona usual y posterior a la marca de negación *sa-*.

El resto de la derivación, incluyendo el comportamiento con respecto a los sufijos de reflexivo y acción recíproca, se modela igual que con las otras personas. En este caso se hace más patente la conveniencia de trabajar con autómatas de dos cabezales, pues de otra manera tendríamos que trabajar con mayor cantidad de estados o agregar mas elementos al léxico para modelar el comportamiento de la derivación verbal con causativos que es un proceso muy productivo en las lenguas chaqueñas.

Para ilustrar el funcionamiento de este dispositivo, consideraremos el siguiente ejemplo:

Ejemplo 4.27 .

qaiqui'aGanataGanaGanaGasomgi: (alguien)le manda dar de comer dentro del agua

$$\overset{que'e,VI}{\rightarrow} q_0 \overset{Caus}{\rightarrow} q_1 \overset{Caus}{\rightarrow} q_2 \overset{Caus}{\rightarrow} q_{14} \overset{Dir}{\rightarrow} q_{17} \overset{Loc}{\rightarrow} q_{18} \overset{PrAc}{\rightarrow} q_{10} \overset{qa-}{\rightarrow} q_{11}$$

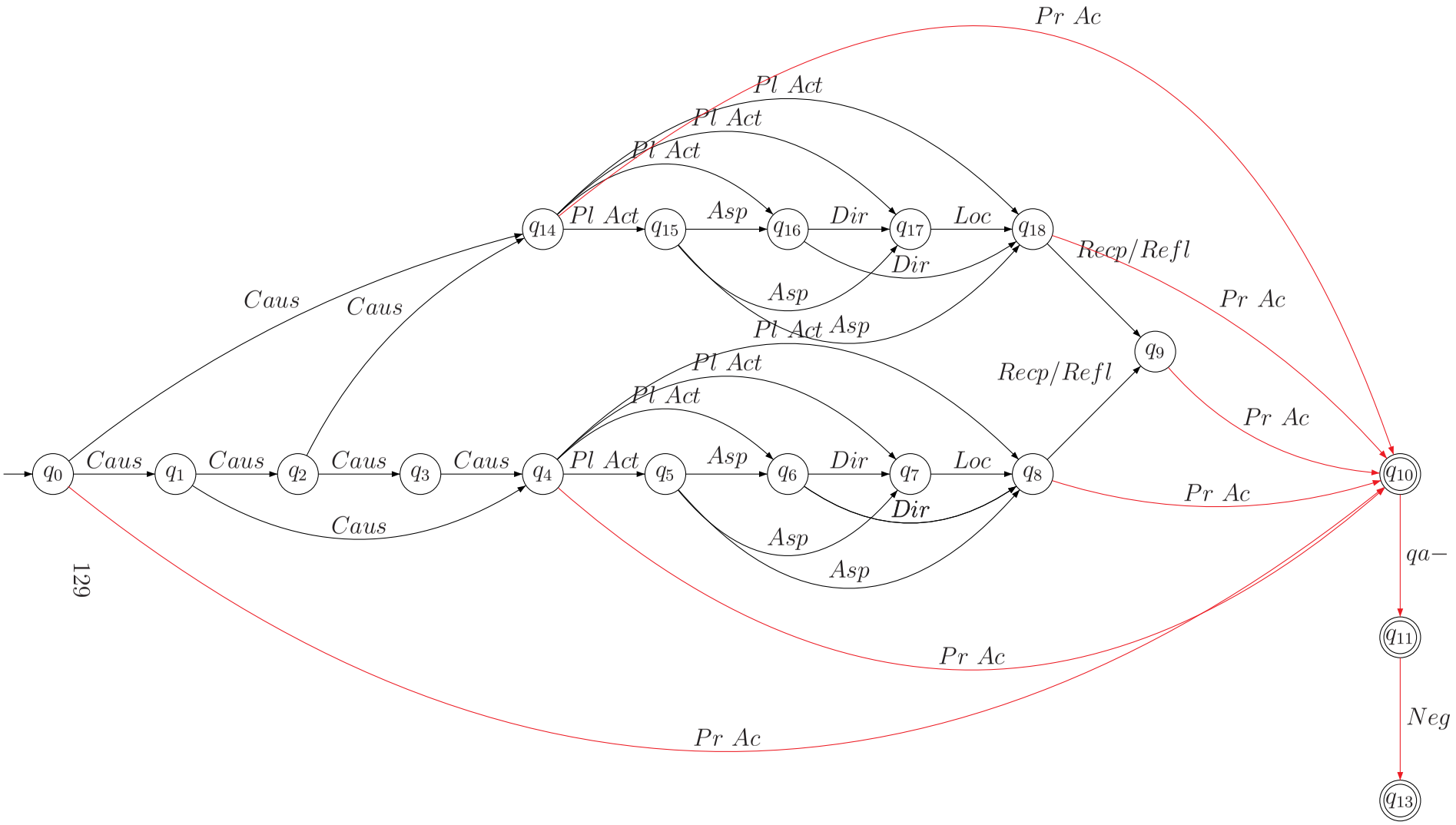


Figura 4.8: Esquema de la morfología verbal del la tercera persona del toba para verbos intransitivos.
Las líneas en negro y rojo marcan las transiciones del cabezal que leen sufijos y prefijos, respectivamente

Apéndice A

Base para KIMMO de la morfología verbal del quechua santiagueño

A.1. Archivo de Reglas en dos niveles

```
;Rules file for Quichua
;last modified Julio 2009


;Andrés Porta                    |  e-mail: hugporta@yahoo.com.ar
;Universidad de Buenos Aires


;CONTENTS
; Medialización de vocales altas
; Elision en sufijos
; END
```

```
;Symbol key:

; + = morpheme break
; # en of morpheme


ALPHABET
  a e i o u p t ch k q s sh ll m n ñ l r w y b d g gg f h x r rr h A E I O U Q Y
  + ’
NULL 0
ANY @
BOUNDARY #
SUBSET C       p t ch k q s sh ll m n ñ l r w y b d f h x rr h Q  ; consonantes
SUBSET V       i e a o u A E I O U                               ; vocales
SUBSET Vlng    I E A O U                                        ; vocales que se el
SUBSET Valt    u i U I                                          ; vocales altas
SUBSET VMed    e o E O                                          ; vocales medias
SUBSET Vbaj    a A
SUBSET Ftr     n y r Y N                          ; fonemas transparentes a la medi
SUBSET Cpos    gg q Q                           ; consonates posvelares
END



RULE "1 Consonant defaults" 1 29

          p t ch k q s sh ll m n ñ l r w y b d g gg f h x r rr h Q W Y
          p t ch k q s sh ll m n ñ l r w y b d g gg f h x r rr h 0 0 y
        1: 1 1 1  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  1 1 1  1 1 1 1



RULE "2 Vowels and other defaults" 1 13
          i e a o u A E I O U N + @
          i e a o u a e i o u n 0 @
```

```
       1: 1 1 1 1 1 1 1 1 1 1 1 1 1


;==========
;Medialización
;==========


RULE "3 Medialización de  i en por postvelar a derecha,
      i:i /<=  __ Ftr:@ CPos:@" 6 5


       i  i   Ftr    CPos  @
       e  i   @      @     @
 1:    2  5   1      1     1
 2:    2  0   3      4     1
 3.    2  5   3      4     1
 4:    2  5   1      1     1
 5:    2  5   6      0     1
 6:    2  5   1      0     1

RULE "4 Medialización de vocal i por postvelar a contigua a izquierda,
      i:i /<= CPos:@ __" " 2 4


     i i  Cpos  @
     e i    @   @
 1:  1 1    2   1
 2:  3 0    2   1
 3:  0 1    2   1



RULE "5 Medialización de u por postvelar a derecha,
      u:u /<=  __ Ftr:@ CPos:@" " 6 5
```

```
      u  u   Ftr    CPos  @
      o  u   @      @     @
 1:   2  5   1      1     1
 2:   2  0   3      4     1
 3.   2  5   3      4     1
 4:   2  5   1      1     1
 5:   2  5   6      0     1
 6:   2  5   1      0     1
```

```
RULE "6 Medialización de vocale u por postvelar a contigua a izquierda,
     u:u /<= CPos:@ __" 3 4
```

```
    u u  Cpos  @
    o u    @   @
 1: 1 1    2   1
 2: 3 0    2   1
 3: 0 1    2   1
```

```
;==========
;Elisión y Epéntesis de fonemas en sufijos
;==========
```

```
RULE "7 epentesis de w en el transición a primera -Wa,
     W:w <=> a:a a:a +:0 ___ a:a +:0 " 9 5


     a  +  W  W  @
     a  0  0  w  @

 1:  2  1  0  1  1
 2:  3  1  0  1  1
 3:  1  4  0  1  1
 4:  2  0  8  5  1
 5.  6  0  0  0  0
 6.  3  7  0  0  0
 7:  2  1  0  1  1
 8:  9  1  1  0  1
 9:  3  0  1  0  1



RULE "8 Elisión de la U en el cislocativo -mU,
      U:0<=> m:m ___+:0 p:p u:u +:0 " 11 6


     +    m    U    U   p    @
     0    m    u    0   p    @

 1:  2    1    1    0   1    1
 2:  2    3    0    0   1    1
 3:  2    0    8    4   0    1
 4.  5    0    0    0   0    0
 5.  0    0    0    0   6    0
 6.  0    0    7    0   0    0
 7:  2    0    0    0   0    0
 8:  9    1    1    1   1    1
```

```
 9:  2    3    1   10   1    1
10:  2    1   11    1   1    1
11:  0    1    1    1   1    1


RULE "9 Elisión de la -n en la marca de tercera pesona después
      del sufijo de pasado, N:0<=> ___+:0 ra +:0 " 12 7


     +    N   N    r    Q    a    @
     0    0   n    r    @    a    @

 1:  2    0   1    1    1    1    1
 2:  0    3   8    1    1    1    1
 3.  4    0   0    0    0    0    0
 4.  0    0   0    5    0    0    0
 5.  0    0   0    0    6    0    0
 6.  0    0   0    0    0    7    0
 7:  1    0   0    0    0    0    0
 8:  9    1   0    1    1    1    1
 9:  2    3   8   10    1    1    1
10:  2    3   8    1   11    1    1
11:  2    8   1    1    1   12    1
12:  0    1   1    1    1    1    1


END
```

A.2. Archivo con la Gramática

```
;quichua.grm
;word grammar for Quichua
;last modified Julio 2009

;This grammar file _requires_ PC-KIMMO version 2.1!!!

;Andrés Porta                        |  e-mail: hugporta@yahoo.com.ar
;Univesidad de Buenos Aires




;=========================================================
;   Parameter settings
;=========================================================

Parameter: Start symbol is Word


;=========================================================
;   Grammar rules
;=========================================================

;Non-terminal symbols: Word
;Terminal symbols: ROOT, SUFFIXTop,SUFFIXGen,SUFFIXCon,SUFFIXAct1Pl, SUFFIXAct2, SUFF


RULE
Word -> Word_1 SUFFIXTop

RULE
Word_2 -> Word_3 SUFFIXGen
```

```
RULE
Word_3 -> Word_4 SUFFIXCon

RULE
Word_4 -> Word_5 SUFFIXAct1Pl|Word_5 SUFFIXAct2|Word_5 SUFFIXAct2Pl|Word_5 SUFFIXAct3

Word_5 -> Word_6 SUFFIXPas

RULE
Word_4 -> Word_6 SUFFIXAct1PlF|Word_6 SUFFIXAct2F|Word_6 SUFFIXAct2PlF|Word_6 SUFFIXA


RULE
Word_6 -> Word_7 SUFFIXModII


RULE
Word_7 -> Word_8 SUFFIXTr1

RULE
Word_8 -> Word_9 SUFFIXMod1

RULE
Word_5 -> Word_10 SUFFIXCaus




RULE
Word_4 -> Word_15 SUFFIXAct1PlF| Word_15 SUFFIXAct2F| Word_15 SUFFIXAct2PlF| Word_15

RULE
Word_15 -> Word_16 SUFFIXPas

RULE
```

```
Word_4 -> Word_16 SUFFIXAct1PlF| Word_16 SUFFIXAct2F| Word_16 SUFFIXAct2PlF| Word_16

RULE
Word_16 -> Word_17 SUFFIXModII


RULE
Word_17 -> Word_10 SUFFIXTr2


RULE
Word_4 -> Word_25 SUFFIXAct1| Word_25 SUFFIXAct1Pl|Word_25 SUFFIXAct2F|Word_25 SUFFIX

RULE
Word_25 -> Word_26 SUFFIXPas

RULE
Word_4 -> Word_26 SUFFIXAct1|Word_26 SUFFIXAct1PlF|Word_26 SUFFIXAct2F|Word_26 SUFFIX

RULE
Word_16 -> Word_17 SUFFIXModII


RULE
Word_17 -> Word_8 SUFFIXTr2


RULE
Word_3 -> Word_34 SUFFIXPl2PO

RULE
Word_33 -> Word_34 SUFFIX2PO

RULE
Word_34 -> Word_5 SUFFIXPas
```

```
RULE
Word_35 -> Word_5 SUFFIXAct1|SUFFIXAct1Pl|Word_25 SUFFIXAct2F|Word_25 SUFFIXAct2Pl|Wo

RULE
Word_25 -> Word_26 SUFFIXPas


RULE
Word_10 -> ROOT

END
```

A.3. Léxico de verbos

```
;verb.lex
;INCLUDE file for quichua.lex
;last modified 07-2009

;Andrés Porta                    |  e-mail: hugporta@yahoo.com.ar
;Universidad de Buenos Aires

;File Contents

;LEXICON V
;  VERBOS


\lf aa
\lx V
\alt SuffixCaus
\gl1 tejer (tr.)
\gl2

\lf achi
\lx V
\alt SuffixCaus
\gl1 v.intr. Estornudar
\gl2

\lf achkayachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Acrecentar, aumentar
\gl2


\lf achkaya
\lx V
```

```
\alt SuffixCaus
\gl1
\gl2 v.intr. Acrecentarse, aumentarse


\lf afanaku
\lx V
\alt SuffixCaus
\gl1 v.intr. (esp) Afanarse
\gl2


\lf agradese
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Agradecer
\gl2


\lf akaku
\lx V
\alt SuffixCaus
\gl1 v.intr. Zurrarse.
\gl2


\lf aka
\lx V
\alt SuffixCaus
\gl1 v.intr. Defecar, cagar. // s. Deposición, deyección.
\gl2


\lf akompaña
\lx V
\alt SuffixCaus
```

\gl1 acompañar
\gl2

\lf
\lx V
\alt SuffixCaus
\gl1
\gl2
Akuychis - v.def. Vamos, vámonos. Se emplea también apocopado: Aku , Akuych y sólo er

\lf allichaku
\lx V
\alt SuffixCaus
\gl1 v.tr. Arreglarse, acicalarse, sanarse
\gl2

\lf allicha
\lx V
\alt SuffixCaus
\gl1 v.tr. Reparar, arreglar, poner en orden, acomodar, adornar, preparar, componer,
asear la casa. // v.tr. Sanar.
\gl2

\lf Alliya
\lx V
\alt SuffixCaus
\gl1 v.tr. Mejorar de salud. Sanar.
\gl2

\lf allpacha
\lx V
\alt SuffixCaus

```
\gl1 v.tr. Polvorear, empolvar.
\gl2



\lf allqocha
\lx V
\alt SuffixCaus
\gl1 v.tr. Burlarse, mofarse, satirizar.
\gl2


\lf allwi
\lx V
\alt SuffixCaus
\gl1 v.tr. Urdir.
\gl2




\lf amka
\lx V
\alt SuffixCaus
\gl1 v.tr. Tostar, torrar.
\gl2



\lf ami
\lx V
\alt SuffixCaus
\gl1
\gl2
```

```
\lf amichi
\lx V
\alt SuffixCaus
\gl1 v.tr. Saciar, llenar.
\gl2


\lf ampi
\lx V
\alt SuffixCaus
\gl1 v.tr. Curar.
\gl2



\lf amu
\lx V
\alt SuffixCaus
\gl1 v.intr. Venir.
\gl2


\lf anaqyachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Elevar.
\gl2



\lf anchu
\lx V
\alt SuffixCaus
\gl1 v.tr. Retirar, apartar.
\gl2
```

```
\lf anukachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Destetar.
\gl2




\lf aña
\lx V
\alt SuffixCaus
\gl1 v.tr. Insultar, injuriar, reprender, blasfemar, denigrar.
\gl2



\lf apamu
\lx V
\alt SuffixCaus
\gl1  v.mov. Traer cosas inanimadas. Portar en dirección hacia el que habla.
\gl2


\lf api
\lx V
\alt SuffixCaus
\gl1 v.tr. Agarrar, capturar, asir, tomar, prender, atrapar, empuñar, coger. // v.tr.
\gl2


\lf apipaku
\lx V
```

```
\alt SuffixCaus
\gl1 Adueñarse, apoderarse, apropiarse
\gl2


\lf apiku
\lx V
\alt SuffixCaus
\gl1 - v.intr. Agarrarse.
\gl2

\lf aqlla
\lx V
\alt SuffixCaus
\gl1 v.tr. Elegir, seleccionar.
\gl2


\lf aqna
\lx V
\alt SuffixCaus
\gl1 v.tr. Heder, apestar. Var. asnay. // s. Tufo.
\gl2



\lf ara
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) - Arar.
\gl2



\lf arka
\lx V
```

```
\alt SuffixCaus
\gl1 v.tr. Atajar, detener, impedir, trancar.
\gl2



\lf arma
\lx V
\alt SuffixCaus
\gl1 v.tr. Bañar
\gl2


\lf armachi
\lx V
\alt Suffix
\gl1 v.tr. Bañar.
\gl2


\lf armaku
\lx V
\alt SuffixCaus
\gl1 v.refl. Bañarse.
\gl2

\lf asi
\lx V
\alt SuffixCaus
\gl1 v.intr. Reirse. // s. Risa.
\gl2


\lf aspi
\lx V
\alt SuffixCaus
```

\gl1 v.tr. Cavar, escarbar.
\gl2

\lf asna
\lx V
\alt SuffixCaus
\gl1 Véase aqnay
\gl2

\lf asta
\lx V
\alt SuffixCaus
\gl1 v.tr. Acarrear.
\gl2

\lf asuti
\lx V
\alt SuffixCaus
\gl1 v.tr. Castigar, azotar, zurrar.
\gl2

\lf atari
\lx V
\alt SuffixCaus
\gl1 v.intr. Levantarse, erguirse.
\gl2

\lf ati
\lx V
\alt SuffixCaus
\gl1 v.tr. Poder, ser capaz de.
\gl2

\lf atichi
\lx V
\alt SuffixC
\gl1 v.tr. Posibilitar.
\gl2

\lf atunya
\lx V
\alt SuffixCaus
\gl1 v.intr. Agrandarse.
\gl2

\lf awi
\lx V
\alt SuffixCaus
\gl1 v.tr. Untar, frotar, embadurnar.
\gl2

\lf ayari
\lx V
\alt SuffixCaus

\gl1 v.intr. Avinagrarse, agriarse.
\gl2

\lf aylluyachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Emparentar.
\gl2

\lf ayqe
\lx V
\alt SuffixCaus
\gl1 v.intr. Huir.
\gl2

\lf ayqechi
\lx V
\alt SuffixCaus
\gl1 v.tr. Ahuyentar.
\gl2

\lf aysa
\lx V
\alt SuffixCaus
\gl1 v.tr. Tirar, jalar, tensar, traccionar, arrastrar. // s. Tracción.
\gl2

\lf aysaku
\lx V
\alt SuffixCaus

\gl1 v.tr. Arrastrarse.
\gl2

\lf ayta
\lx V
\alt SuffixCaus
\gl1 v.tr. Patear, dar coces. // s. Patada.
\gl2

\lf bale
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Valer, costar.
\gl2

\lf bawtisa
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Bautizar.
\gl2

\lf bellakya
\lx V
\alt SuffixCaus
\gl1 v.int. (reg.) Encabritarse.
\gl2

\lf bolyaku
\lx V
\alt SuffixCaus
\gl1 v.int. (esp) Volver, regresar, retornar.

```
\gl2


\lf bolyachi
\lx V
\alt SuffixCaus
\gl1  v.tr. (esp) Retornar.
\gl2

\lf chawa
\lx V
\alt SuffixCaus
\gl1 v.tr. Ordeñar, exprimir. Var. chaay
\gl2



\lf chaki
\lx V
\alt SuffixCaus
\gl1 v.intr. Secar, perder humedad. // s. Sed // s. Sequía. // v.intr. Tener sed..
\gl2


\lf chakichi
\lx V
\alt SuffixCaus
\gl1 v.tr. Desecar, deshidratar. Secar, hacer perder humedad.
\gl2


\lf chakiku
\lx V
\alt SuffixCaus
\gl1 v.intr. Secarse.
\gl2
```

```
\lf challway
\lx V
\alt SuffixCaus
\gl1 v.tr. Pescar.
\gl2


\lf chamka
\lx V
\alt SuffixCaus
\gl1 v.tr. Tocar, palpar.
\gl2


\lf chamqa
\lx V
\alt SuffixCaus
\gl1 v.tr. Triturar, quebrantar, cascar.
\gl2


\lf chanpa
\lx V
\alt SuffixCaus
\gl1 v.tr. Obstruir un curso de agua mediante champas. // v.tr. Colocar césped.
\gl2


\lf chaplla
\lx V
\alt SuffixCaus
\gl1 - v.tr. Probar la comida
\gl2
```

```
\lf chapre
\lx V
\alt SuffixCaus
\gl1 v.tr. Sacudir, agitar.
\gl2


\lf chaqchi
\lx V
\alt SuffixCaus
\gl1 v.tr. Rociar. // s. Aspersión.
\gl2


\lf chaqru
\lx V
\alt SuffixCaus
\gl1 v.tr. Mezclar, entreverar.
\gl2


\lf chaski
\lx V
\alt SuffixCaus
\gl1 v.tr. Quitar, arrebatar. // v.tr. Recibir, aceptar.
\gl2



\lf chawpiya
\lx V
\alt SuffixCaus
\gl1 v.tr. Reducirse una cosa a la mitad.
\gl2
```

```
\lf chaya
\lx V
\alt SuffixCaus
\gl1 v.intr. Llegar, alcanzar un punto determinado, arribar. // v.intr. Cocerse los a
\gl2


\lf chee
\lx V
\alt SuffixCaus
\gl1 v.tr. Exprimir.
\gl2


\lf cheqni
\lx V
\alt SuffixCaus
\gl1 v.tr. Aborrecer, despreciar. // s. Encono.
\gl2


\lf cheqaya
\lx V
\alt SuffixCaus
\gl1 v.intr. Enderezarse.
\gl2


\lf cheqayachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Enderezar
\gl2
```

```
\lf chichuchi
\lx V
\alt SuffixCaus
\gl1 v.tr. Embarazar, empreñar, preñar.
\gl2


\lf chilpa
\lx V
\alt SuffixCaus
\gl1 v.tr. Curar aplicando la chilpa.
\gl2


\lf chimpa
\lx V
\alt SuffixCaus
\gl1 v.tr. Vadear un curso de agua. Entrar al agua y caminar por ella.
\gl2


\lf chinka
\lx V
\alt SuffixCaus
\gl1
\gl2 v.intr. Perderse, desaparecer. // v.intr. Aturdirse, ofuscarse. Chinkachiy - v.t

\lf chinkachi
\lx V
\alt SuffixCaus
\gl1
\gl2 v.tr. Extraviar, desorientar.

\lf chipa
\lx V
\alt SuffixCaus
```

```
\gl1 v.tr. Apretar, prensar.
\gl2


\lf chiri
\lx V
\alt SuffixCaus
\gl1 v.intr. Hacer frío.
\gl2


\lf chiriya
\lx V
\alt SuffixCaus
\gl1 v.intr. Enfriarse.
\gl2


\lf chiriyachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Enfriar.
\gl2


\lf chisiyachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Hacer demorar algo hasta el atardecer.
\gl2


\lf chisiya
\lx V
\alt SuffixCaus
\gl1 v.intr. Atardecer.
```

```
\gl2


\lf chisi
\lx V
\alt SuffixCaus
\gl1 v.intr. Tardecer.
\gl2


\lf chiya
\lx V
\alt SuffixCaus
\gl1 v.tr. Espulgar para sacar las liendres.
\gl2


\lf choqa
\lx V
\alt SuffixCaus
\gl1 v.tr. Tirar, arrojar, lanzar.
\gl2


\lf chonqa
\lx V
\alt SuffixCaus
\gl1 v.tr. Chupar. // v.tr. Absorber, sorber.
\gl2

\lf chukcha
\lx V
\alt SuffixCaus
\gl1 v.tr. Tirar de los cabellos.
\gl2
```

```
\lf chukukiya
\lx V
\alt SuffixCaus
\gl1 v.int. Temblar, tiritar.
\gl2


\lf chuma
\lx V
\alt SuffixCaus
\gl1 v.tr. Escurrir.
\gl2


\lf chumalichi
\lx V
\alt SuffixCaus
\gl1 v.tr. Azuzar al perro.
\gl2


\lf chuña
\lx V
\alt SuffixCaus
\gl1 v.intr. Vomitar.
\gl2


\lf chupuya
\lx V
\alt SuffixCaus
\gl1 v.intr. Formarse un absceso.
\gl2
```

\lf chura
\lx V
\alt SuffixCaus
\gl1 v.tr. Poner, colocar, depositar, ubicar, posar.
\gl2

\lf churaku
\lx V
\alt SuffixCaus
\gl1 v.tr. Ponerse.
\gl2

\lf chusa
\lx V
\alt SuffixCaus
\gl1 v.intr. Faltar, ser insuficiente, no alcanzar. // v.intr. Ausentarse.
\gl2

\lf chuskiya
\lx V
\alt SuffixCaus
\gl1 v.tr. Mostrar vanidad. // v.tr. Despreciar las comidas.
\gl2

\lf chusuyachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Ahuecar.
\gl2

\lf chusuya
\lx V

```
\alt SuffixCaus
\gl1 v.intr. Ahuecarse.
\gl2


\lf chusya
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp.) Chuzar, apuñalar.
\gl2



\lf chutki
\lx V
\alt SuffixCaus
\gl1 v.tr. Zafar, dislocar.
\gl2

\lf chuyancha
\lx V
\alt SuffixCaus
\gl1 v.tr. Enjuagar
\gl2


\lf chuyaya
\lx V
\alt SuffixCaus
\gl1 v.intr. Diluirse.
\gl2


\lf chuyayachi
\lx V
\alt SuffixCaus
```

```
\gl1 v.tr. Diluir, aclarar un líquido.
\gl2



\lf dansa
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp)Bailar.
\gl2


\lf dañuchi
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Dañar.
\gl2



\lf desya
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Desear.
\gl2


\lf dewdaku
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Adeudar.
\gl2


\lf dewe
\lx V
```

```
\alt SuffixCaus
\gl1 v.tr. (esp) Deber, adeudar.
\gl2


\lf eduka
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Educar.
\gl2


\lf eqe
\lx V
\alt SuffixCaus
\gl1 v.tr. Ahogar, asfixiar.
\gl2


\lf eqeku
\lx V
\alt SuffixCaus
\gl1 v.intr. Ahogarse, asfixiarse.
\gl2



\lf eskribi
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Escribir.
\gl2



\lf estudya
```

```
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Estudiar.
\gl2


\lf gana
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Ganar.
\gl2


\lf gasta
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Gastar.
\gl2


\lf gatya
\lx V
\alt SuffixCaus
\gl1 v.tr. (reg.) Arrastrarse para subrepticiamente introducirse en el lecho de una m
\gl2


\lf gusta
\lx V
\alt SuffixCaus
\gl1 v.intr. (esp) Gustar, agradar.
\gl2


\lf icha
\lx V
```

```
\alt SuffixCaus
\gl1 v.tr. (esp) Echar, derramar, verter.
\gl2


\lf ichaku
\lx V
\alt SuffixCaus
\gl1 v.intr. Derramarse.
\gl2



\lf ichu
\lx V
\alt SuffixCaus
\gl1 v. tr. Segar. // s. Siega.
\gl2


\lf iki
\lx V
\alt SuffixCaus
\gl1 v.tr. Copular. // s. Coito.
\gl2


\lf illu
\lx V
\alt SuffixCaus
\gl1 v.tr. Enfermar de deseo, de antojo.
\gl2


\lf importa
\lx V
```

```
\alt SuffixCaus
\gl1 v.tr. (esp) Importar.
\gl2


\lf ina
\lx V
\alt SuffixCaus
\gl1 v.tr. Hacer (referido a cosas inmateriales). Obrar, actuar, proceder.
\gl2


\lf inaku
\lx V
\alt SuffixCaus
\gl1 v.tr. Hacerse, tornarse, volverse, transformarse, imitar, aparentar.
\gl2


\lf ismu
\lx V
\alt SuffixCaus
\gl1 v.intr. Descomponer, entrar en putrefacción.
\gl2



\lf ishpa
\lx V
\alt SuffixCaus
\gl1 v.tr. Orinar.
\gl2


\lf kacha
\lx V
```

```
\alt SuffixCaus
\gl1 v.tr. Mandar, enviar, emitir.
\gl2



\lf kachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Salar, poner sal.
\gl2


\lf kallpacha
\lx V
\alt SuffixCaus
\gl1 v.tr. Dar fuerzas, alentar.
\gl2



\lf kallpachaku
\lx V
\alt SuffixCaus
\gl1 v.intr. Esforzarse.
\gl2


\lf kama
\lx V
\alt SuffixCaus
\gl1 v.intr. Caber.
\gl2
```

```
\lf kamachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Adaptar.
\gl2



\lf kambiya
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Cambiar, trocar.
\gl2



\lf kami
\lx V
\alt SuffixCaus
\gl1 v.tr. Injuriar, ridiculizar, satirizar.
\gl2



\lf kanka
\lx V
\alt SuffixCaus
\gl1 v.tr. Asar.
\gl2



\lf kanchachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Encender la lumbre.
```

```
\gl2



\lf kancha
\lx V
\alt SuffixCaus
\gl1 v.tr. Alumbrar. // v.intr. Brillar, iluminar.
\gl2



\lf kani
\lx V
\alt SuffixCaus
\gl1 v.tr. Morder, dentellar. Picar un insecto. // s. Mordisco, tarascón.
\gl2



\lf kanta
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Cantar.
\gl2



\lf kanti
\lx V
\alt SuffixCaus
\gl1 v.tr. Torcer. // s. Torsión.
\gl2
```

```
\lf kapa
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Capar, castrar.
\gl2



\lf kara
\lx V
\alt SuffixCaus
\gl1 v.tr. Picar, arder. // s. Picazón, ardor.
\gl2



\lf karuncha
\lx V
\alt SuffixCaus
\gl1 v.tr. Alejar, aislar.
\gl2



\lf karunchaku
\lx V
\alt SuffixCaus
\gl1 v.intr. Alejarse, aislarse.
\gl2



\lf karuyachi
\lx V
\alt SuffixCaus
```

\gl1 v.tr. Alejar, distanciar, aislar, apartar.
\gl2

\lf karuya
\lx V
\alt SuffixCaus
\gl1 v.intr. Alejarse, distanciarse.
\gl2

\lf kasachi
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Cazar.
\gl2

\lf kasaraku
\lx V
\alt SuffixCaus
\gl1 v.intr. (esp) Casarse, contraer matrimonio, desposar. // s. (esp) Boda, casamien
\gl2

\lf kasu
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Obedecer.
\gl2

```
\lf kawsa
\lx V
\alt SuffixCaus
\gl1 v.intr. Vivir . // s. Vida.
\gl2



\lf kawsari
\lx V
\alt SuffixCaus
\gl1 v.intr. Revivir.
\gl2



\lf ka
\lx V
\alt SuffixCaus
\gl1 v.cop. Ser, haber.
\gl2



\lf kichari
\lx V
\alt SuffixCaus
\gl1 v.tr. Abrir.
\gl2



\lf kichki
\lx V
\alt SuffixCaus
```

\gl1 v.tr. Atorar, estreñir, estrangular.
\gl2

\lf killinsa
\lx V
\alt SuffixCaus
\gl1 v.tr. Carbonizar
\gl2

\lf kincha
\lx V
\alt SuffixCaus
\gl1 v.tr. Construir paredes de ramas o cañas.
\gl2

\lf kipu
\lx V
\alt SuffixCaus
\gl1 v.tr. Anudar.
\gl2

\lf kontesta
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Contestar.
\gl2

```
\lf kreyi
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Creer. Var. kriyiy [kri:y], kreey [kre:y].
\gl2



\lf krusa
\lx V
\alt SuffixCaus
\gl1 v.mov. (esp) Cruzar, atravesar.
\gl2



\lf kuchu
\lx V
\alt SuffixCaus
\gl1 v.tr. Cortar, talar, tajar, trocear, sesgar, tronchar, truncar, trozar, podar, a
\gl2



\lf kuchuku
\lx V
\alt SuffixCaus
\gl1 v.tr. Cortarse.
\gl2



\lf kumuku
\lx V
\alt SuffixCaus
```

```
\gl1 v.refl. Inclinarse, agacharse. Var. kumuykuy.
\gl2



\lf kuruya
\lx V
\alt SuffixCaus
\gl1 v.intr. Agusanarse
\gl2



\lf kururu
\lx V
\alt SuffixCaus
\gl1 v.tr. Ovillar.
\gl2



\lf kusichi
\lx V
\alt SuffixCaus
\gl1 v.tr. Alegrar
\gl2



\lf kusiku
\lx V
\alt SuffixCaus
\gl1 v.intr. Alegrarse, alborozarse, contentarse. //s. Felicidad
\gl2
```

```
\lf kuta
\lx V
\alt SuffixCaus
\gl1 v.tr. Moler, triturar.
\gl2
```

```
\lf kutipa
\lx V
\alt SuffixCaus
\gl1 v.intr. Rumiar.
\gl2
```

```
\lf kuti
\lx V
\alt SuffixCaus
\gl1 v.intr. Quedar, sobrar.
\gl2
```

```
\lf kutichi
\lx V
\alt SuffixCaus
\gl1 v.tr. Retener, hacer quedar, no devolver.
\gl2
```

```
\lf kutiku
\lx V
\alt SuffixCaus
\gl1 v.intr. Quedarse, permanecer.
\gl2



\lf kuya
\lx V
\alt SuffixCaus
\gl1 v.tr. Tener lástima.
\gl2



\lf kuyaku
\lx V
\alt SuffixCaus
\gl1 v.tr. Compadecerse, apiadarse.
\gl2



\lf kuyu
\lx V
\alt SuffixCaus
\gl1 v.tr. Mover, torcer. // s. Movimiento.
\gl2


\lf kuyurichi
\lx V
\alt SuffixCaus
\gl1 v.tr. Poner en movimiento, activar
\gl2
```

```
\lf kuyuri
\lx V
\alt SuffixCaus
\gl1 v.intr. Moverse, ponerse en movimiento.
\gl2



\lf kwenta
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Contar, relatar
\gl2



\lf kwida
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Cuidar.
\gl2



;-----------------------------------------------------------------------------



\lf lasa
\lx V
\alt SuffixCaus
\gl1 v.tr. Pesar, acción de la gravedad. // s. Peso, pesadez.
\gl2
```

```
\lf lawra
\lx V
\alt SuffixCaus
\gl1 v.tr. Arder el fuego o algo combustible. // s. Combustión.
\gl2


\lf lawrachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Encender fuego.
\gl2


\lf lirya
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Discutir, pelear, lidiar.
\gl2


\lf lokoya
\lx V
\alt SuffixCaus
\gl1 v.intr. (esp) Enloquecer. // s. Demencia.
\gl2



\lf lulu
\lx V
\alt SuffixCaus
\gl1 v.tr. Acariciar. //s. Caricia.
\gl2
```

```
\lf llaki
\lx V
\alt SuffixCaus
\gl1 v.intr. Tener pena.
\gl2



\lf llakichi
\lx V
\alt SuffixCaus
\gl1 v.tr. Apenar, entristecer, angustiar, afligir (a alguien).
\gl2


\lf llakiku
\lx V
\alt SuffixCaus
\gl1 v.intr. Apenarse, entristecerse, afligirse, angustiarse.
\gl2



\lf llalli
\lx V
\alt SuffixCaus
\gl1 v.tr. Pasar, aventajar, ganar, adelantarse.
\gl2


\lf
\lx V
\alt SuffixCaus
```

```
\gl1
\gl2
Llamkay - v.tr. Trabajar. // s. Trabajo, labor, tarea.


\lf
\lx V
\alt SuffixCaus
\gl1
\gl2
Llampuyachiy - v.tr. Ablandar.


\lf
\lx V
\alt SuffixCaus
\gl1
\gl2
Llampuyay - v.intr. Ablandarse.


\lf
\lx V
\alt SuffixCaus
\gl1
\gl2
Llatanakuy - v.tr. Desnudarse, desvestirse.


\lf llatana
\lx V
\alt SuffixCaus
\gl1 v.tr. Desnudar, desvestir. Var. llantanay.
\gl2
```

\lf llantu
\lx V
\alt SuffixCaus
\gl1 v.tr. Sombrear.
\gl2

\lf llañuyachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Adelgazar, disminuir el diámetro de un cuerpo largo o cilíndrico. Angostar
\gl2

\lfllañuya
\lx V
\alt SuffixCaus
\gl1 v.intr. Adelgazarse.
\gl2

\lf llapsayachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Adelgazar, disminuir el espesor.
\gl2

\lf llapsaya

\lx V
\alt SuffixCaus
\gl1 v.intr. Adelgazarse, disminuirse el espesor.
\gl2

\lf llaqwa
\lx V
\alt SuffixCaus
\gl1 v.tr. Lamer.
\gl2

\lf llawsa
\lx V
\alt SuffixCaus
\gl1 v.intr. Babear, segregar viscosidad.
\gl2

\lf llikcha
\lx V
\alt SuffixCaus
\gl1 v.tr. Despertarse.
\gl2

\lf llikchachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Despertar.
\gl2

```
\lf llikcha
\lx V
\alt SuffixCaus
\gl1 v.tr. Asemejar.
\gl2


\lf llikchaku
\lx V
\alt SuffixCaus
\gl1 v.intr. Parecerse, asemejarse. // s. Similitud.
\gl2



\lf lliki
\lx V
\alt SuffixCaus
\gl1 v.tr. Romper, desgarrar, destrozar, desvirgar.
\gl2



\lf llilli
\lx V
\alt SuffixCaus
\gl1 v.tr. Irritarse la comisura de los labios. // s. Afta . Boquera.
\gl2



\lf llipipiya
\lx V
```

```
\alt SuffixCaus
\gl1 v.intr. Relampaguear, destellar, titilar, brillar, centellear.
\gl2



\lf lloqa
\lx V
\alt SuffixCaus
\gl1 v.tr. Trepar, subir, ascender, montar. Aparear animales.
\gl2


\lf lloqsi
\lx V
\alt SuffixCaus
\gl1 v.intr. Salir, partir.
\gl2



\lf llota
\lx V
\alt SuffixCaus
\gl1 v.tr. Colmar, llenar, atestar.
\gl2


\lf lluchu
\lx V
\alt SuffixCaus
\gl1 v.tr. Desollar, despellejar, cuerear.
\gl2
```

```
\lf llulla
\lx V
\alt SuffixCaus
\gl1 v.tr. Mentir, engañar
\gl2



\lf llunchi
\lx V
\alt SuffixCaus
\gl1  v.tr. Pulir.
\gl2


\lf lluspiyachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Pulir, alisar.
\gl2






\lf machachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Embriagar.
\gl2



\lf macha
\lx V
\alt SuffixCaus
```

\gl1 v.tr. Emborracharse , embriagarse.
\gl2

\lf machuya
\lx V
\alt SuffixCaus
\gl1 v.int. Envejecer, avejentarse.
\gl2

\lf malli
\lx V
\alt SuffixCaus
\gl1 v.tr. Probar el sabor, gustar, saborear.
\gl2

\lf mancha
\lx V
\alt SuffixCaus
\gl1 v.intr. Temer. // s. Miedo, temor, terror, pánico, pavor.
\gl2

\lf manchachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Acobardar, asustar, amedrentar, atemorizar, aterrorizar.
\gl2

```
\lf manchaku
\lx V
\alt SuffixCaus
\gl1 v.intr. Asustarse, atemorizarse.
\gl2



\lf mansuyachi
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Amansar.
\gl2


\lf manta
\lx V
\alt SuffixCaus
\gl1 v.tr. Tender, extender.
\gl2



\lf mantari
\lx V
\alt SuffixCaus
\gl1 v.tr. Desplegar una cosa extendiéndola.
\gl2




\lf maña
\lx V
\alt SuffixCaus
```

```
\gl1 v.tr. Pedir, pretender, demandar. // v.tr. Prestar.
\gl2
```

```
\lf mapacha
\lx V
\alt SuffixCaus
\gl1 v.tr. Ensuciar, tiznar.
\gl2
```

```
\lf maqa
\lx V
\alt SuffixCaus
\gl1 v.tr. Pegar, castigar, agredir, vapulear, aporrear. // s. Paliza.
\gl2
```

```
\lf maqanya
\lx V
\alt SuffixCaus
\gl1 v.tr. Dar palos, apalear.
\gl2
```

```
\lf maqolla
\lx V
\alt SuffixCaus
\gl1 v.intr. (esp) Retoñar.
\gl2
```

```
\lf maqsi
\lx V
\alt SuffixCaus
\gl1 v.intr. Cabecear dormitando.
\gl2



\lf marcha
\lx V
\alt SuffixCaus
\gl1 v.mov. (esp) Marchar, caminar.
\gl2



\lf marqa
\lx V
\alt SuffixCaus
\gl1 v.tr. Traer o llevar una cosa entre los brazos. // v.tr. Abrazar.
\gl2



\lf maska
\lx V
\alt SuffixCaus
\gl1 v.tr. Buscar.
\gl2


\lf matya
\lx V
\alt SuffixCaus
\gl1 v.tr. Matear, tomar mate, desayunar.
```

```
\gl2


\lf mawkaya
\lx V
\alt SuffixCaus
\gl1 v.intr. Envejecer, caer en desuso.
\gl2


\lf maylla
\lx V
\alt SuffixCaus
\gl1  v.tr. Lavar cosas no absorbentes. // v.tr. Asear el cuerpo.
\gl2



\lf micha
\lx V
\alt SuffixCaus
\gl1  v.tr. Mezquinar. //s. Miseria.
\gl2



\lf miri
\lx V
\alt SuffixCaus
\gl1 v.intr. (esp) Merecer.
\gl2


\lf miku
\lx V
\alt SuffixCaus
```

\gl1 v.tr. Comer, almorzar, alimentarse. // s. Comida, alimento. // s. Almuerzo, cena
\gl2

\lf mikuchi
\lx V
\alt SuffixCaus
\gl1 v.tr. Alimentar.
\gl2

\lf mikunaya
\lx V
\alt SuffixCaus
\gl1 v.intr. Tener deseos de comer. // s. Apetito.
\gl2

\lf millachi
\lx V
\alt SuffixCaus
\gl1 v.intr. Repugnar, asquear.
\gl2

\lf milla
\lx V
\alt SuffixCaus
\gl1 v.intr. Tener asco.
\gl2

\lf millpu

\lx V
\alt SuffixCaus
\gl1 v.tr. Tragar, deglutir, ingerir, devorar, engullir.
\gl2

\lf minka
\lx V
\alt SuffixCaus
\gl1 v.tr. Encargar, encomendar un trabajo, pedir un servicio.
\gl2

\lf mira
\lx V
\alt SuffixCaus
\gl1 v.intr. Reproducirse el ganado en forma natural.
\gl2

\lf mirachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Hacer reproducir el ganado.
\gl2

\lf mishkiyachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Endulzar, dulcificar.
\gl2

```
\lf moqchiku
\lx V
\alt SuffixCaus
\gl1 v.intr. Enjuagarse la boca.
\gl2



\lf mosqo
\lx V
\alt SuffixCaus
\gl1 v.intr. Soñar. // s. Sueño, pesadilla.
\gl2



\lf motochi
\lx V
\alt SuffixCaus
\gl1 v.tr. Amputar, mutilar, tronchar.
\gl2



\lf muchanaku
\lx V
\alt SuffixCaus
\gl1 v.recip. Besarse.
\gl2



\lf mucha
\lx V
\alt SuffixCaus
```

```
\gl1 v.tr. Besar.
\gl2
```

```
\lf muchu
\lx V
\alt SuffixCaus
\gl1 v.intr. Carecer. // s. Carestía, necesidad, pobreza.
\gl2
```

```
\lf muku
\lx V
\alt SuffixCaus
\gl1 v.tr. Mascar, masticar.
\gl2
```

```
\lf muna
\lx V
\alt SuffixCaus
\gl1 v.tr. Querer, amar, desear, pretender, anhelar, ansiar, codiciar, apetecer. // s
\gl2
```

```
\lf munaku
\lx V
\alt SuffixCaus
\gl1 v.intr. Amarse.
\gl2
```

```
\lf munanaku
\lx V
\alt SuffixCaus
\gl1 v.recip. Amarse, enamorarse.
\gl2



\lf mutki
\lx V
\alt SuffixCaus
\gl1 v.tr. Oler, aspirar.
\gl2



\lf muyu
\lx V
\alt SuffixCaus
\gl1 v.intr. Girar, rotar, dar vueltas.
\gl2


\lf nanachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Herir, lastimar, torturar, lacerar, ultrajar, provocar dolor.
\gl2



\lf nana
\lx V
\alt SuffixCaus
```

\gl1 v.intr. Doler.
\gl2

\lf ni
\lx V
\alt SuffixCaus
\gl1 v.tr. Decir.
\gl2

\lf ñaka
\lx V
\alt SuffixCaus
\gl1 v.tr. Pegar duro y sostenido.
\gl2

\lf ñaqcha
\lx V
\alt SuffixCaus
\gl1 v.tr. Peinar.
\gl2

\lf ñaqchaku
\lx V
\alt SuffixCaus
\gl1 v.tr. Peinarse.
\gl2

\lf ñawpa

\lx V
\alt SuffixCaus
\gl1 v.tr. e intr. Adelantar, adelantarse, aventajar, anticipar, preceder.
\gl2

\lf ñeqe
\lx V
\alt SuffixCaus
\gl1 v.tr. Abollar, hundir o quebrar de un golpe.
\gl2

\lf ñiti
\lx V
\alt SuffixCaus
\gl1 v.tr. Apretar, prensar, oprimir, presionar, compactar, comprimir. // s. Presión.
\gl2

\lf ñukña
\lx V
\alt SuffixCaus
\gl1 v.tr. Presumir, cortejar, galantear, coquetear.
\gl2

\lf ñuñuchi
\lx V
\alt SuffixCaus
\gl1 v.tr. Amamantar.
\gl2

```
\lf ñuñu
\lx V
\alt SuffixCaus
\gl1 v.tr. Mamar.
\gl2


\lf ñutu
\lx V
\alt SuffixCaus
\gl1 v.tr. Aplastar, compactar.
\gl2



\lf onqo
\lx V
\alt SuffixCaus
\gl1  v.intr. Enfermar. // s. Enfermedad, peste.
\gl2



\lf onqoku
\lx V
\alt SuffixCaus
\gl1  v.intr. Enfermarse.
\gl2




\lf oqlla
\lx V
\alt SuffixCaus
\gl1 v.tr. Empollar, incubar.
```

```
\gl2


\lf oqo
\lx V
\alt SuffixCaus
\gl1 v.tr. Mojar, empapar, humedecer.
\gl2




\lf oqochi
\lx V
\alt SuffixCaus
\gl1 v.tr. Salpicar, mojar.
\gl2


\lf oqoku
\lx V
\alt SuffixCaus
\gl1 v.tr. Mojarse, empaparse.
\gl2



;-------------------------------------------------------------------------------


\lf paa
\lx V
\alt SuffixCaus
\gl1 v.tr. Volar. // s. Vuelo. Paaq - adj. Volador.
\gl2
```

\lf pachalliku
\lx V
\alt SuffixCaus
\gl1 v.intr. Vestirse, cubrirse de ropas.
\gl2

\lf pagara
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp). Pagar, saldar una deuda.
\gl2

\lf paka
\lx V
\alt SuffixCaus
\gl1 v.tr. Ocultar, esconder, encubrir. Pakakuy - v.intr. Esconderse, ocultarse, tapa
\gl2

\lf paki
\lx V
\alt SuffixCaus
\gl1 v.tr. Romper, quebrar, fracturar, quebrantar, partir, cascar, destrozar. Paki- s
\gl2

\lf pallankya
\lx V
\alt SuffixCaus
\gl1 v.tr. Enlazar y sacar un animal de entre un grupo de animales.

```
\gl2




\lf palla
\lx V
\alt SuffixCaus
\gl1  v.tr. Juntar, recoger, recolectar, cosechar, colectar. Pallana - s. Juego infar
\gl2


\lf panta
\lx V
\alt SuffixCaus
\gl1 v.tr. Equivocarse, desconocer, confundir. Pantakuy - v.intr. Confundirse.
\gl2




\lf paqari
\lx V
\alt SuffixCaus
\gl1 v.intr. Amanecer. // s. Madrugada, amanecer, alba, alborada, aurora.
\gl2



\lf paqcha
\lx V
\alt SuffixCaus
\gl1 v.tr. Poner boca abajo un recipiente. Volcar.
\gl2
```

```
\lf paqkacha
\lx V
\alt SuffixCaus
\gl1 v.tr. Transmitir una cosa la mala suerte.
\gl2



\lf paruya
\lx V
\alt SuffixCaus
\gl1 v.intr. Tostarse por la aplicación de un fuego demasiado vivo y repentino.
\gl2


\lf paska
\lx V
\alt SuffixCaus
\gl1 v.tr. Desatar. Paskana - adv.l. Lugar donde se desata los animales. En los Valle
\gl2


\lfpaspa
\lx V
\alt SuffixCaus
\gl1  v.intr. Resecarse la piel formándose escamas.
\gl2


\lf pasya
\lx V
\alt SuffixCaus
\gl1 v.intr. (esp) Pasear.
\gl2
```

\lf patara
\lx V
\alt SuffixCaus
\gl1 v.tr. Desdoblar, extender una cosa al sol para que se seque.
\gl2

\lf pensa
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Pensar.
\gl2

\lf picha
\lx V
\alt SuffixCaus
\gl1 v.tr. Barrer. // v.tr. Asear, limpiar. Pichakuy - v.intr. Limpiarse, despejarse
\gl2

\lf pikaniya
\lx V
\alt SuffixCaus
\gl1 v.tr. Picanear.
\gl2

\lf pinki

```
\lx V
\alt SuffixCaus
\gl1  v.intr. Saltar, brincar. // s. Salto.
\gl2


\lf pintu
\lx V
\alt SuffixCaus
\gl1  v.tr. Envolver, vendar.
\gl2



\lf pipiluchi
\lx V
\alt SuffixCaus
\gl1 v.tr. Desorejar.
\gl2



\lf pishi
\lx V
\alt SuffixCaus
\gl1 v.intr. Faltar, escasear, no ser suficiente. Pishiyachiy - v.tr. Disminuir, redu
\gl2



\lf pita
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Fumar.
\gl2
```

\lf piti
\lx V
\alt SuffixCaus
\gl1 v.tr. Cortar (un hilo, una cuerda, una soga).
\gl2

\lf pitiya
\lx V
\alt SuffixCaus
\gl1 v.intr. Cortarse (una cuerda, una corriente de agua, la vida).
\gl2

\lf posoqe
\lx V
\alt SuffixCaus
\gl1 v.intr. Espumar, echar espuma una cosa.
\gl2

\lf posqo
\lx V
\alt SuffixCaus
\gl1 v.intr. Podrirse una cosa. Var. poshqoy. // s. Podredumbre. Posqochiy - v.tr. P
\gl2

\lf puklla

```
\lx V
\alt SuffixCaus
\gl1 v.tr. Jugar. // s. Juego, deporte.
\gl2


\lf punki
\lx V
\alt SuffixCaus
\gl1 v.tr. Hincharse, inflamarse, abultarse. // s. Inflamación, tumefacción, edema.
\gl2


\lf puñu
\lx V
\alt SuffixCaus
\gl1 v.intr. Dormir. Puñuchiy - v.tr. Adormecer. Puñulu - adj. Dormilón. Puñuna - s.
\gl2


\lf puri
\lx V
\alt SuffixCaus
\gl1 v.mov. Andar, caminar, transitar, vagar, vagabundear, peregrinar, deambular.
\gl2



\lf pusa
\lx V
\alt SuffixCaus
\gl1 v.tr. Llevar, conducir, transportar personas o animales vivos.
\gl2
```

```
\lf pushka
\lx V
\alt SuffixCaus
\gl1 v.tr. Hilar. Pushkadora - s. Hilandera. Pushkana - s. Huso.
\gl2
```

```
\lf puyu
\lx V
\alt SuffixCaus
\gl1 v.intr. Nublarse.
\gl2
```

```
\lf qaa
\lx V
\alt SuffixCaus
\gl1 v.tr. Ver, mirar, atisbar. // s. Vista, visión.
\gl2
```

```
\lf qaachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Mostrar, aclarar un asunto, demostrar.
\gl2
```

```
\lf qaaku
\lx V
\alt SuffixCaus
```

\gl1 v.tr. Mirarse. Qaana - s. Mirador.
\gl2

\lf qachiri
\lx V
\alt SuffixCaus
\gl1 v.tr. Escarbar la tierra las aves. Escarbar. Desordenar.
\gl2
y -

\lf qallaqchi
\lx V
\alt SuffixCaus
\gl1 v.tr. Rejuntar restos de cosecha entre los rastrojos.
\gl2

\lf qallari
\lx V
\alt SuffixCaus
\gl1 v.tr. Comenzar, empezar, principiar. // s. Principio.
\gl2

\lf qapari
\lx V
\alt SuffixCaus
\gl1 v.tr. Gritar, vocear, vociferar. Por extensión: bramar. // s. Grito, alarido, br
\gl2

```
\lf qaqo
\lx V
\alt SuffixCaus
\gl1 v.tr. Refregar, frotar, amasar.
\gl2



\lf qasa
\lx V
\alt SuffixCaus
\gl1 v.intr. Helar, nevar, congelar.
\gl2



\lf qaspa
\lx V
\alt SuffixCaus
\gl1 v.tr. Chamuscar. Asar.
\gl2



\lf qata
\lx V
\alt SuffixCaus
\gl1  v.tr. Cubrir, tapar, abrigar, cobijar. Qatakuna - s. Manta, abrigo, sábana, cob
\gl2



\lf qati
\lx V
\alt SuffixCaus
```

\gl1 v.tr. Perseguir, arrear, ahuyentar.
\gl2

\lf qea
\lx V
\alt SuffixCaus
\gl1 v.tr. Drenar pus.
\gl2

\lfqecha
\lx V
\alt SuffixCaus
\gl1 v.tr. Evacuar líquido diarreico.
\gl2

\lf qechu
\lx V
\alt SuffixCaus
\gl1 v.tr. Quitar, despojar, arrebatar.
\gl2

\lf qee
\lx V
\alt SuffixCaus
\gl1 v.tr. Torcer. // s. Torsión.
\gl2

\lf qemi
\lx V
\alt SuffixCaus
\gl1 v.tr. Arrimar, acercar. // s. Acercamiento. Qemichiy - v.tr. Arrimar, acercar, a
\gl2

\lf qenti
\lx V
\alt SuffixCaus
\gl1 v.intr. Encogerse, contraerse.
\gl2

\lf qentichi
\lx V
\alt SuffixCaus
\gl1 v.tr. Encoger, contraer.
\gl2

\lf qepa
\lx V
\alt SuffixCaus
\gl1 v.intr. Dar la planta frutos tardíos.
\gl2

\lf qeshifriya
\lx V
\alt SuffixCaus
\gl1 v.intr. Pestañear.

\gl2

\lf qeshpi
\lx V
\alt SuffixCaus
\gl1 v.intr. Librarse, libertarse. Escaparse, fugarse, salvarse de un peligro. Qeshp
\gl2

\lf
\lx V
\alt SuffixCaus
\gl1
\gl2
qochpay - v.tr. Revolcar.

\lf qonqa
\lx V
\alt SuffixCaus
\gl1 v.tr. Olvidar.
\gl2

\lf qonqoriku
\lx V
\alt SuffixCaus
\gl1 v.intr. Arrodillarse, postrarse.
\gl2

\lf qora

\lx V
\alt SuffixCaus
\gl1 v.tr. Escardar, carpir, cortar la hierba, desmalezar.
\gl2

\lf qo
\lx V
\alt SuffixCaus
\gl1 v.tr. Dar, entregar.
\gl2

\lf ranti
\lx V
\alt SuffixCaus
\gl1 v.tr. Comprar, adquirir.
\gl2

\lf rantiku
\lx V
\alt SuffixCaus
\gl1 v.tr. Vender. // s. Venta.
\gl2

\lf reqsi
\lx V
\alt SuffixCaus
\gl1 v.tr. Conocer, reconocer. Reqsinakuy - v.recip. Conocerse. Reqsisqa - adj. Conoc
\gl2

```
\lf rikuri
\lx V
\alt SuffixCaus
\gl1  v.intr. Aparecer, reaparecer, asomarse.
\gl2
```

```
\lf rima
\lx V
\alt SuffixCaus
\gl1 v.intr. Hablar. // s. Habla, palabra, charla. Rimachiy - v.tr. Conversar. Rimaq
\gl2
```

```
\lf ri
\lx V
\alt SuffixCaus
\gl1 v.mov. Ir, acudir, partir, asistir, concurrir.
\gl2
```

```
\lf rumiya
\lx V
\alt SuffixCaus
\gl1 v.intr. Petrificarse.
\gl2
```

\lf runtu
\lx V
\alt SuffixCaus
\gl1 v.intr. Poner huevos, aovar, desovar.
\gl2

\lf rupa
\lx V
\alt SuffixCaus
\gl1 v.intr. Hacer calor, quemar el sol. // v.tr. Quemar. // s. Calor. Rupachiy - v.
\gl2

\lf rutu
\lx V
\alt SuffixCaus
\gl1 s. Esquilar, tusar, trasquilar. Pelar. Por extensión cortar el cabello a una pe
\gl2

\lf ruwa
\lx V
\alt SuffixCaus
\gl1 v.tr. Hacer, realizar, efectuar, ejecutar, construir, elaborar. Ruwanayoq - adj.
\gl2

\lf rrakuya
\lx V
\alt SuffixCaus
\gl1 v.intr. Engrosarse.
\gl2

```
\lf saksa
\lx V
\alt SuffixCaus
\gl1 v.intr. Saciarse, comer hasta el hartazgo. // s. Saciedad.
\gl2



\lf sama
\lx V
\alt SuffixCaus
\gl1 v.int. Respirar. // s. Aliento.
\gl2



\lf sama
\lx V
\alt SuffixCaus
\gl1 v.int. Descansar. // s. Descanso.
\gl2



\lf saqma
\lx V
\alt SuffixCaus
\gl1 v.tr. Abofetear, dar puñetazos.
\gl2


\lf saqmanaku
```

```
\lx V
\alt SuffixCaus
\gl1  v.recip. Boxear.
\gl2


\lf saqrayachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Afear.
\gl2



\lf saqe
\lx V
\alt SuffixCaus
\gl1 v.tr. Dejar, abandonar, desechar.
\gl2


\lf saru
\lx V
\alt SuffixCaus
\gl1  v.tr. Pisar, apisonar. // s. Pisada. // v.intr. Aparearse las aves.
\gl2


\lf sati
\lx V
\alt SuffixCaus
\gl1 v.tr. Meter, introducir, encajar.
\gl2
```

```
\lf saya
\lx V
\alt SuffixCaus
\gl1 v.tr. Parar, detener. Sayakuy - v.tr. Pararse, detenerse. Sayana - s. Parada, pa
\gl2



\lf sayku
\lx V
\alt SuffixCaus
\gl1 v.intr. Cansarse, fatigarse. // s. Cansancio.
\gl2


\lf saykuchi
\lx V
\alt SuffixCaus
\gl1 v.t. Cansar.
\gl2



\lf sebya
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Cebar.
\gl2


\lf sekya
\lx V
\alt SuffixCaus
\gl1  v.tr. Fumar tragando el humo.
\gl2
```

```
\lf sera
\lx V
\alt SuffixCaus
\gl1 v.tr. Coser, confeccionar.
\gl2


\lf sikincha
\lx V
\alt SuffixCaus
\gl1 v.tr. Escardar las plantas arrimando tierra a la base.
\gl2




\lf sillu
\lx V
\alt SuffixCaus
\gl1 v.tr. Pellizcar, uñar, arañar.
\gl2



\lf simpa
\lx V
\alt SuffixCaus
\gl1 v.tr. Trenzar.
\gl2


\lf sipi
\lx V
```

```
\alt SuffixCaus
\gl1 v.tr. Estrangular, ceñir, constreñir.
\gl2



\lf sipriya
\lx V
\alt SuffixCaus
\gl1 v.intr. Reírse a carcajadas.
\gl2



\lf sipu
\lx V
\alt SuffixCaus
\gl1 v.tr. Fruncir, arrugar.
\gl2



\lf siri
\lx V
\alt SuffixCaus
\gl1 v.intr. Estar acostado, yacer. Sireq - adj. Yaciente. Sirichiy - v.tr. Acostar.
\gl2



\lf sirwi
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Servir. // s. Servicio.
\gl2
```

\lf sisaya
\lx V
\alt SuffixCaus
\gl1 v.intr. Florecer.
\gl2

\lf sitki
\lx V
\alt SuffixCaus
\gl1 v.tr. Expeler ventosidad por el ano sin ruido. Peer.
\gl2

\lf soqari
\lx V
\alt SuffixCaus
\gl1 v.tr. Levantar, alzar.
\gl2

\lf sorqo
\lx V
\alt SuffixCaus
\gl1 v.tr. Sacar, extraer, arrancar, desenterrar.
\gl2

\lf sukllayachi
\lx V

```
\alt SuffixCaus
\gl1 v.tr. Unificar, aunar.
\gl2



\lf sullu
\lx V
\alt SuffixCaus
\gl1 v.intr. Abortar.
\gl2




\lf suniyachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Alargar, estirar, ahondar, profundizar.
\gl2



\lf suniya
\lx V
\alt SuffixCaus
\gl1 v.intr. Alargarse, estirarse, ahondarse, profundizarse.
\gl2



\lf sustuku
\lx V
\alt SuffixCaus
\gl1 v.intr. (esp) Asustarse.
\gl2
```

\lf susu
\lx V
\alt SuffixCaus
\gl1 v.tr. Cernir, tamizar, zarandear, colar.
\gl2

\lf sutichi
\lx V
\alt SuffixCaus
\gl1 v.tr. Denominar.
\gl2

\lf sutu
\lx V
\alt SuffixCaus
\gl1 v.intr. Gotear.Suwa - s. Ladrón. Suway - v.tr. Robar, saquear.
\gl2

\lf suya
\lx V
\alt SuffixCaus
\gl1 v.tr. Esperar, aguardar.
\gl2

\lf sheqshi
\lx V
\alt SuffixCaus
\gl1 v.impers. Escocer, dar comezón, picar.

\gl2
y -

\lf shintakuy
\lx V
\alt SuffixCaus
\gl1 v.intr. Tartamudear.
\gl2

\lf taki
\lx V
\alt SuffixCaus
\gl1 v.tr. (des.) Cantar.
\gl2

\lf talli
\lx V
\alt SuffixCaus
\gl1 v.tr. Vaciar, verter, trasvasar.
\gl2

\lf tanqa
\lx V
\alt SuffixCaus
\gl1 v.tr. Empujar.
\gl2

\lf tapu
\lx V
\alt SuffixCaus
\gl1 v.tr. Preguntar, averiguar. // s. Pregunta.
\gl2

\lf taqwe
\lx V
\alt SuffixCaus
\gl1 v.tr. Mecer, mezclar la comida. Taqwena - s. Mezclador.
\gl2

\lf taqsa
\lx V
\alt SuffixCaus
\gl1 v.tr. Lavar una cosa absorbente.
\gl2

\lf taripa
\lx V
\alt SuffixCaus
\gl1 v.intr. Alcanzar.
\gl2

\lf tari
\lx V

\alt SuffixCaus
\gl1 v.tr. Hallar, encontrar una cosa.
\gl2

\lf tariku
\lx V
\alt SuffixCaus
\gl1 v.intr. Encontrarse. // v.tr. Encontrarlos.
\gl2

\lf tarpu
\lx V
\alt SuffixCaus
\gl1 v.tr. Sembrar. // s. Siembra. Tarpoq - s. Sembrador, agricultor. Tarpuy pacha -
\gl2
y

\lf tartanchiya
\lx V
\alt SuffixCaus
\gl1 v.intr. Tartamudear.
\gl2

\lf tikiya
\lx V
\alt SuffixCaus
\gl1 v.tr. Arrojar cascotes, apedrear. // s. Pedrea.
\gl2

```
\lf tikra
\lx V
\alt SuffixCaus
\gl1 v.tr. Voltear, revertir.
\gl2



\lf tikraku
\lx V
\alt SuffixCaus
\gl1 v.tr. Virar.
\gl2



\lf tinka
\lx V
\alt SuffixCaus
\gl1 v.tr. Dar capirotazos.
\gl2



\lf tinkuku
\lx V
\alt SuffixCaus
\gl1 v.tr. Confluir. Tinkulu - adj. Patizambo.
\gl2


\lf tinkunaku
\lx V
\alt SuffixCaus
```

\gl1 v.recip. Encontrarse dos o más personas. Toparse. Converger.
\gl2

\lf tinku
\lx V
\alt SuffixCaus
\gl1 v.intr. Encontrarse, reunirse (dos personas, animales o cosas).
\gl2

\lf tinpu
\lx V
\alt SuffixCaus
\gl1 v.intr. Hervir. // s. Ebullición. Tinpuchiy - v.tr. Hacer hervir.
\gl2
y -

\lf tipi
\lx V
\alt SuffixCaus
\gl1 v.tr. Cosechar, deshojar la mazorca del maíz.
\gl2

\lf tisa
\lx V
\alt SuffixCaus
\gl1 v.tr. Cardar la lana. // v.intr. Desgastarse una tela por el uso.
\gl2

\lf tishpi
\lx V
\alt SuffixCaus
\gl1 v.tr. Picotear, picar. // v.tr. Pellizcar. // s. Picotazo.
\gl2

\lf toqa
\lx V
\alt SuffixCaus
\gl1 v. tr. Salivar, escupir.
\gl2

\lf toqe
\lx V
\alt SuffixCaus
\gl1 v.intr. Transpirar, sudar.
\gl2

\lf toqlla
\lx V
\alt SuffixCaus
\gl1 v.tr. Enlazar.
\gl2

\lf toqya
\lx V
\alt SuffixCaus
\gl1 v.intr. Reventar, explotar, explosionar.

\gl2

\lf tuku
\lx V
\alt SuffixCaus
\gl1 v.tr. Terminar, acabar, agotar.
\gl2

\lf tukuku
\lx V
\alt SuffixCaus
\gl1 v.intr. Terminarse, acabarse, agotarse, perecer.
\gl2

\lf tuksi
\lx V
\alt SuffixCaus
\gl1 v.tr. Hincar, punzar, pinchar, aguijonear, picar. // s. Pinchadura, punción, pir
\gl2

\lf tulluya
\lx V
\alt SuffixCaus
\gl1 v.intr. Adelgazar, enflaquecer una persona o animal, demacrarse. Tulluyachiy - v
\gl2

\lf tumpa
\lx V
\alt SuffixCaus
\gl1 v.tr. Acusar, culpar, achacar, atribuir.

```
\gl2


\lf tuñi
\lx V
\alt SuffixCaus
\gl1 v.tr. Derrumbarse, desplomarse, desmoronarse.
\gl2


\lf tuñichi
\lx V
\alt SuffixCaus
\gl1 v.tr. Derribar, derrumbar, desplomar, desmoronar, demoler.
\gl2



\lf tupu
\lx V
\alt SuffixCaus
\gl1 v.tr. Medir, comparar, mensurar.
\gl2



\lf tusa
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Tusar, trasquilar, rapar.
\gl2



\lf tusu
\lx V
```

```
\alt SuffixCaus
\gl1 v.tr. (arc.) Bailar, danzar. Voz reemplazada por el hispanismo dansay. Tusuq - s
\gl2



\lf uchalliku
\lx V
\alt SuffixCaus
\gl1 v.tr. Delinquir, pecar.
\gl2



\lf uhu
\lx V
\alt SuffixCaus
\gl1 v.tr. onom.Toser.
\gl2



\lf ullpu
\lx V
\alt SuffixCaus
\gl1 v.tr. Comer harina ullpu. Var. ullpyay.
\gl2



\lf umpi
\lx V
\alt SuffixCaus
\gl1 v.intr. Sudar, transpirar
\gl2
```

```
\lf umpu
\lx V
\alt SuffixCaus
\gl1 v.intr. Encoger.
\gl2


\lf umpuya
\lx V
\alt SuffixCaus
\gl1 v.intr. Encogerse, acurrucarse.
\gl2



\lf umu
\lx V
\alt SuffixCaus
\gl1 v.tr. Embrujar, brujear.
\gl2




\lf unancha
\lx V
\alt SuffixCaus
\gl1 v.intr. Entender, comprender, interpretar.
\gl2



\lf una
\lx V
\alt SuffixCaus
```

```
\gl1 v.int. Tardar demorar, retrasar.
\gl2


\lf unayachi
\lx V
\alt SuffixCaus
\gl1 v.tr.Hacer retrasar.
\gl2


\lf unaya
\lx V
\alt SuffixCaus
\gl1 v.intr. Demorarse, retrasarse
\gl2



\lf unta
\lx V
\alt SuffixCaus
\gl1 v.intr. Llenarse, colmarse. // s. Saciedad.
\gl2


\lf untachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Llenar, colmar, saciar. Por extensión: embarazar.
\gl2


\lf unuyachi
\lx V
\alt SuffixCaus
```

```
\gl1 v.tr. Derretir, licuar, aguar.
\gl2


\lf unuya
\lx V
\alt SuffixCaus
\gl1 v.intr. Derretirse, licuarse, aguarse.
\gl2



\lf upallachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Acallar, silenciar.
\gl2


\lf upalla
\lx V
\alt SuffixCaus
\gl1 v.intr. Callarse, enmudecer.
\gl2


\lf upalliya
\lx V
\alt SuffixCaus
\gl1 v.tr. Pegar de callado, a traición.
\gl2


\lf upaya
\lx V
\alt SuffixCaus
```

\gl1 v.intr. Ensordecer, perder la audición.
\gl2

\lf upya
\lx V
\alt SuffixCaus
\gl1 v.tr. Beber.
\gl2

\lf upta
\lx V
\alt SuffixCaus
\gl1 v.tr. Fermentar.
\gl2

\lf urayku
\lx V
\alt SuffixCaus
\gl1 v.intr. Bajarse, descender, apearse, descabalgar.
\gl2

\lf urma
\lx V
\alt SuffixCaus
\gl1 v.intr. Caer.
\gl2

```
\lf urmachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Abatir, tumbar, tirar, volcar, voltear.
\gl2


\lf urmaku
\lx V
\alt SuffixCaus
\gl1 v.tr. Tumbarse.
\gl2

\lf urmanaya
\lx V
\alt SuffixCaus
\gl1 v.intr. Tambalear.
\gl2


\lf usachaku
\lx V
\alt SuffixCaus
\gl1 v.intr. Infestarse de piojos.
\gl2


\lf usaku
\lx V
\alt SuffixCaus
\gl1 v.tr. Despiojarse. Usasapa
\gl2


\lf usa
\lx V
```

```
\alt SuffixCaus
\gl1 v.tr. Espulgar, despiojar.
\gl2


\lf usu
\lx V
\alt SuffixCaus
\gl1 v.intr. Carecer. // s. Carestía, necesidad, pobreza, miseria.
\gl2


\lf ushpacha
\lx V
\alt SuffixCaus
\gl1 v.tr. Ensuciar, cubrir con cenizas.
\gl2


\lf utki
\lx V
\alt SuffixCaus
\gl1 v.intr. Hundirse, zambullirse.
\gl2


\lf utkichi
\lx V
\alt SuffixCaus
\gl1 v.tr. Hundir, zambullir. Utkikoq - adj. Zambullidor.
\gl2



\lf utku
\lx V
```

\alt SuffixCaus
\gl1 v.tr. Agujerear, perforar.
\gl2

\lf utqa
\lx V
\alt SuffixCaus
\gl1 v.intr. Apurarse, apresurarse. // s. Premura.
\gl2

\lf utulayachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Acortar, disminuir, achicar, empequeñecer.
\gl2

\lf utulaya
\lx V
\alt SuffixCaus
\gl1 v.intr. Acortarse, disminuirse, achicarse, decrecer.
\gl2

\lf uyari
\lx V
\alt SuffixCaus
\gl1 v.tr. Oir, escuchar, sentir.
\gl2

```
\lf uywa
\lx V
\alt SuffixCaus
\gl1 v.tr. Criar niños o animales domésticos.
\gl2


\lf wacha
\lx V
\alt SuffixCaus
\gl1
\gl2 v.intr. Parir, dar a luz. // s. Parto.


\lf wachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Flechar, aguijonear, pinchar.
\gl2


\lf wakchaya
\lx V
\alt SuffixCaus
\gl1 v.intr. Empobrecer.
\gl2


\lf wakchiya
\lx V
\alt SuffixCaus
\gl1 v.intr. Tener un hijo la mujer después de mucho tiempo.
\gl2
```

\lf wana
\lx V
\alt SuffixCaus
\gl1 v. intr. Escarmentar. // s. Escarmiento.
\gl2

\lf wanchi
\lx V
\alt SuffixCaus
\gl1 v.tr. Matar. Se aplica a animales pequeños e insectos . También significa apagar
\gl2

\lf wanya
\lx V
\alt SuffixCaus
\gl1 v.intr. Bostear.
\gl2

\lf wañu
\lx V
\alt SuffixCaus
\gl1 v.intr. Morir, fallecer.
\gl2

\lf wañuchi
\lx V
\alt SuffixCaus
\gl1 v.tr. Asesinar, matar (seres humanos y animales de gran tamaño)
\gl2

```
\lf wañuna
\lx V
\alt SuffixCaus
\gl1 v.intr. Morir, fallecer.
\gl2

\lf wañuri
\lx V
\alt SuffixCaus
\gl1 v.intr. Agonizar. // v. intr. Entumecer
\gl2.

\lf waqa
\lx V
\alt SuffixCaus
\gl1 v.intr. Llorar, plañir.
\gl2



\lf waqaycha
\lx V
\alt SuffixCaus
\gl1 v.tr. Guardar, ahorrar, preservar, atesorar.
\gl2


\lf waqlli
\lx V
\alt SuffixCaus
\gl1 v.tr. Descomponer, echar a perder. Decolorar, descolorir, despintar, desteñir. A
\gl2
```

\lf waqta
\lx V
\alt SuffixCaus
\gl1 v.tr. Golpear, dar golpes, tañer. Ejecutar instrumentos musicales de cuerda o pe
\gl2

\lf waqtari
\lx V
\alt SuffixCaus
\gl1 v.tr. Golpetear, tamborilear.
\gl2

\lf waqya
\lx V
\alt SuffixCaus
\gl1 v.tr. Llamar.
\gl2

\lf warku
\lx V
\alt SuffixCaus
\gl1 v.tr. Colgar, ahorcar.
\gl2

\lf warkuku
\lx V
\alt SuffixCaus
\gl1 v.intr. Pender. Warkuna - s. Colgadero.
\gl2

```
\lf waskya
\lx V
\alt SuffixCaus
\gl1 v.tr. Azotar, castigar con lonja.
\gl2



\lf wata
\lx V
\alt SuffixCaus
\gl1 v.tr. (esp) Atar, amarrar.
\gl2



\lf watuku
\lx V
\alt SuffixCaus
\gl1 v.tr. Extrañar, sentir nostalgia.
\gl2


\lf watya
\lx V
\alt SuffixCaus
\gl1 v.tr. Asar con piedras calientes y bajo tierra.
\gl2



\lf wawqeya
\lx V
\alt SuffixCaus
\gl1 v.tr. Confraternizar.
```

```
\gl2


\lf wawqechanaku
\lx V
\alt SuffixCaus
\gl1 v.recip. Hermanarse.
\gl2


\lf  wayrachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Ventilar, aventar. Wayramuyu - s. Remolino, torbellino. Wayray - v.intr. C
\gl2


\lf wayrakacha
\lx V
\alt SuffixCaus
\gl1 v.mov. Correr.
\gl2


\lf wayta
\lx V
\alt SuffixCaus
\gl1 v.mov. Nadar.
\gl2



\lf weqe
\lx V
\alt SuffixCaus
\gl1 v.intr. Lagrimear.
```

\gl2

\lf wichka
\lx V
\alt SuffixCaus
\gl1 v.tr. Cerrar, trancar, taponar, clausurar, encerrar.
\gl2

\lf wikchu
\lx V
\alt SuffixCaus
\gl1 v.tr. Regurgitar. // v.tr. Echar, arrojar, abandonar, botar, desechar.
\gl2

\lf willa
\lx V
\alt SuffixCaus
\gl1 v.tr. Avisar, advertir, prevenir, presagiar, predecir, anunciar, comunicar, cont
\gl2

\lf wiña
\lx V
\alt SuffixCaus
\gl1 v.intr. Crecer, criarse.
\gl2

\lf wishi

\lx V
\alt SuffixCaus
\gl1 v.tr. Sacar líquido de un recipiente.
\gl2

\lf wishipu
\lx V
\alt SuffixCaus
\gl1 v.tr. Trasvasar líquido de un recipiente a otro, trasegar. //s. Transfusión.
\gl2

\lf yaarcha
\lx V
\alt SuffixCaus
\gl1 v.tr. Ensangrentar
\gl2

\lf yachaku
\lx V
\alt SuffixCaus
\gl1 v.intr. Acostumbrarse, aclimatarse, adaptarse.
\gl2

\lf yachapiya
\lx V
\alt SuffixCaus
\gl1 v.tr. Imitar
\gl2

\lf yacha
\lx V

```
\alt SuffixCaus
\gl1 v.tr. Saber. // s. Sabiduría.
\gl2



\lf yanapa
\lx V
\alt SuffixCaus
\gl1 v.tr. Ayudar, colaborar. // Auxiliar. Cooperar.
\gl2



\lf yanapanaku
\lx V
\alt SuffixCaus
\gl1 v.recip. Ayudarse mutuamente.
\gl2



\lf yanu
\lx V
\alt SuffixCaus
\gl1 v.tr. Cocinar.
\gl2



\lf yapa
\lx V
\alt SuffixCaus
\gl1 v.tr. Añadir, suplementar, agregar, adicionar, unir, aumentar, conectar.
\gl2
```

```
\lf yarqa
\lx V
\alt SuffixCaus
\gl1 v.intr. Tener hambre. // s. Hambre. Apetito.
\gl2
```

```
\lfyayku
\lx V
\alt SuffixCaus
\gl1 v.int. Entrar, adentrarse, penetrar. Yaykuna - s. Entrada, acceso. Yayqoq - s. E
\gl2
```

```
\lf yoqo
\lx V
\alt SuffixCaus
\gl1 v.tr. Copular.
\gl2
```

```
\lf yupa
\lx V
\alt SuffixCaus
\gl1 v.tr. Contar, enumerar, computar. // s. Cuenta, cómputo.
\gl2
```

```
\lf yuraqyachi
\lx V
\alt SuffixCaus
\gl1 v.tr. Blanquear.
\gl2



\lf yuraqya
\lx V
\alt SuffixCaus
\gl1 v.intr. Blanquearse, encanecer.
\gl2



\lf yuya
\lx V
\alt SuffixCaus
\gl1 v.tr. Recordar, memorar. // s. Memoria.
\gl2


\lf yuyari
\lx V
\alt SuffixCaus
\gl1 v.intr. Acordarse, conmemorar.
\gl2



\lf
\lx V
\alt SuffixCaus
\gl1
```

```
\gl2



;END OF FILE
```

A.4. Léxico de sufijos

```
;affix.lex
;INCLUDE file for quichua.lex
;last modified Julio-2009

;Andrés Porta                    |  e-mail: hugporta@yahoo.com.ar
;Universidad de Buenos Aires     |


;LEXICON INITIAL

\lf 0
\lx INITIAL
\alt Verb
\gl1
\gl2

\lf 0
\lx INITIAL
\alt Prefix
\gl1
\gl2



; SUFIJOS MODALES

\lf +Pa
\lx SUFFIXModI
\alt SuffixTr
\fea
\gl1
\gl2

\lf +ri
```

```
\lx SUFFIXModI
\alt SuffixTr
\fea
\gl1
\gl2

\lf +Ia
\lx SUFFIXModI
\alt SuffixTr
\fea
\gl1
\gl2

\lf +na
\lx SUFFIXModI
\alt SuffixTr
\fea
\gl1
\gl2

\lf +chi
\lx SUFFIXModI
\alt SuffixTr
\fea
\gl1
\gl2

\lf +mU
\lx SUFFIXModI
\alt SuffixTr
\fea
\gl1
\gl2

\lf +ku
\lx SUFFIXModI
```

```
\alt SuffixTr
\fea
\gl1
\gl2

;SUFIJOS PERSONAS OBJETO

\lf +pU
\lx SUFFIXTr3
\alt SuffixModII
\fea
\gl1 tercera objeto
\gl2

\lf +wA
\lx SUFFIXTr1
\alt SuffixModII
\fea
\gl1 primera objeto
\gl2

\lf +su
\lx SUFFIXTr2
\alt SuffixModII
\fea
\gl1
\gl2

;SUFIJOS MODALES II

\lf +naya
\lx SUFFIXModII
\alt SuffixPas
\fea
\gl1
\gl2
```

```
\lf +na
\lx SUFFIXModII
\alt SuffixPas
\fea
\gl1
\gl2

\lf +chka
\lx SUFFIXModII
\alt SuffixPas
\fea
\gl1
\gl2

;SUFIJOS TEMPORALES

\lf +ra
\lx SUFFIXPas
\alt SuffixAg
\fea
\gl1
\gl2

;SUFIJOS PERSONALES ACTOR

\lf +ni
\lx SUFFIXAg1
\alt SuffixCond
\fea
\gl1
\gl2


\lf +Y
\lx SUFFIXAg2
```

```
\alt SuffixCond
\fea
\gl1
\gl2


\lf +n
\lx SUFFIXAg3
\alt SuffixCond
\fea
\gl1
\gl2


\lf +nki
\lx SUFFIX
\alt SuffixCaus
\fea
\gl1
\gl2


\lf +sqa
\lx SUFFIXAgFut1
\alt SuffixCond
\fea
\gl1
\gl2

\lf +sa
\lx SUFFIXFut2
\alt SuffixCond
\fea
\gl1
\gl2
```

```
\lf +saj
\lx SUFFIXFut3
\alt SuffixCaus
\fea
\gl1
\gl2

\lf +nqR
\lx SUFFIXFutPl2
\alt SuffixCaus
\fea
\gl1
\gl2




;SUFIJOS OBJETO II

\lf +ki
\lx SUFFIX
\alt SuffixCaus
\fea
\gl1
\gl2

\lf +chis
\lx SUFFIX
\alt SuffixCaus
\fea
\gl1
\gl2


;SUFIJO CONDICIONAL
```

```
\lf +man
\lx SUFFIXCond
\alt SuffixGen
\fea
\gl1
\gl2

;SUFIJOS GENERALES

\lf +lla
\lx SUFFIXGen
\alt SuffixTop
\fea
\gl1
\gl2

\lf +pas
\lx SUFFIXGen
\alt SuffixTop
\fea
\gl1
\gl2

\lf +pis
\lx SUFFIXGen
\alt SuffixTop
\fea
\gl1
\gl2

\lf +taj
\lx SUFFIXGen
\alt SuffixTop
\fea
\gl1
\gl2
```

```
\lf +qa
\lx SUFFIXTop
\alt
\fea
\gl1
\gl2


\lf +chu
\lx SUFFIXGen
\alt SuffixTop
\fea
\gl1
\gl2

\lf +mi
\lx SUFFIXGen
\alt SuffixTop
\fea
\gl1
\gl2

\lf +si
\lx SUFFIXGen
\alt SuffixTop
\fea
\gl1
\gl2



\lf +cha
\lx SUFFIXGen
\alt SuffixTop
\fea
```

```
\gl1
\gl2

\lf +chus
\lx SUFFIXGen
\alt SuffixTop
\fea
\gl1
\gl2
```

Bibliografía

[1] Albarracín, Lelia Inés, Mario C. Tebes y Jorge Alderetes (eds.) 2002. Introducción al quichua santiagueño por Ricardo L.J. Nardi. Buenos Aires: Editorial DUNKEN.

[2] Alderetes, Jorge Ricardo. 1997. Vocabulario quichua-castellano. http://webs.satlink.com/usuarios/r/rory/vocabu.htm.

[3] Alderetes, Jorge Ricardo.2001.El quichua de Santiago del Estero. Gramática y vocabulario. Tucumán: Facultad de Filosofía y Letras, UNT.

[4] Antworth, Evan L. 1990. PC-KIMMO: a two-level processor for morphological analysis.No. 16 in Occasional publications in academic computing. Dallas: Summer Institute of Linguistics.

[5] Barton, E. 1986. Computational Complexity in Two-Level Morphology, in 24th Annual Meeting of the Association for Computational Linguistics.

[6] Barton, E. 1987. Berwick, R, and Ristaxl, E.,The Complexity of Two-Level Morphology, chapter 5, in Computational Complexity and Natural Language, MIT Press, Cambridge, MA.

[7] Barton, E., R. Berwick, E. Ristad. 1987. Computational Complexity and Natural Language. The MIT Press, Cambridge MA.

[8] Beesley, Kenneth R. and Lauri Karttunen. 2003. Finite State Morphology. PaloAlto, CA: CSLI Publications.

[9] Bravo Domingo A. 1965. Estado actual del quichua santiagueño. Tucumán, Argentina: Universidad Nacional de Tucumán.

[10] Bravo, Domingo A. 1987. Diccionario Castellano-Quichua Santiagueño. Santiago del Estero, Argentina: Ediciones Kelka, El Liberal.

[11] Bravo Domingo A. 1987. Diccionario Castellano-Quichua Santiagueño. Santiago del Estero, Argentina: Ediciones Kelka, El Liberal.

[12] Buckwalter, Alberto. 1995. Vocabulario Mocoví, Mennonite Board of Missions, Elkhart, Indiana.

[13] Buckwalter, Alberto. 2001 Vocabulario toba. Formosa / Indiana, Equipo Menonita.

[14] Censabella, Marisa. 1999. Las lenguas indígenas de Argentina. Una mirada actual. EUDEBA, Buenos Aires.

[15] Cerron-Palomino, Rodolfo.1987. Lingüística Quechua. Cusco, Perú: Centro de Estudios Rurales Andinos Bartolomé de las Casas.

[16] Chomsky, Noam. 1957. Syntactic structures. The Hague: Mouton.

[17] Chomsky, Noam. 1965. On certain formal properties of grammars. In R. D. Luce, R. R. Bush, and E.Galanter, eds., Readings in mathematical psychology, Vol. II, 323-418. New York: John Wiley and Sons. First published in 1959.

[18] Chomsky, Noam and Morris Halle. 1968. The sound pattern of English. New York: Harper and Row.

[19] Creider, Chet, Hankamer, Jorge y Wood, Derick. 1995. Preset two-head automata and morphological analysis of natural language. International Journal of Computer Mathematics, 1029-0265, Volume 58, Issue 1, pp. 1-18.

[20] Eilenberg, Samuel. 1974-1976. Automata, languages and machines, 2 vols. Academic press.

[21] Garey, M. R. and D. S. Johnson. 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman.

[22] Gesrzestein, A. 1994. Lengua maká. Estudio descriptivo. Buenos Aires: Instituto de Lingüística, Facultad de Filosofía y Letras, U.B.A. (Colección "Nuestra América". Serie Archivo de lenguas Indoamericanas.

[23] Gregores, Emma y Jorge A. Suárez. 1967. A description of colloquial Guaraní. La Haya: Mouton.

[24] Guasch, Antonio, S.J. 1978. El idioma guaraní. Gramática y antología de prosa y verso. Asunción: Ediciones Loyola.

[25] Gussenhoven,Carlos y Haike Jacobs.1998. Understanding phonology. London: Arnold. Co-published in New York: Oxford University Press.

[26] Heinz, Joos. 1973. Einteilung der Laute im Neuosmanischen auf der Grundlage des türkischen Alphabets. Manuscript University Zurich.

[27] Heinz, Joos. 1984. Porque los matemáticos no hablan turco, VII Coloquio Nacional de Algebra, Córdoba, Argentina, 6-17.VIII-1984.

[28] Heinz, Joos. 1988. 31st Permanent International Conference (PIAC), Weimar,GDR, 14-17.6.88: Zum Problem de Kontextfreiheit de türkischen Morphologie.

[29] Heinz, Joos y Claus Schönig. 1991. Turcic Morphology as Regular Language, Central Asiatic Journal, 1-2, pp 96-122.

[30] Hopcroft J. and Ullman J. 1979. Introduction to Automata Theory, Languages and Computatuion. Addison-Wesley.

[31] Hunt, R.J. 1940. Mataco grammar. Revisada por el Rev. B.A. Tompkins. Tucumán, Instituto de Antropología.

[32] Johnson, C. Douglas. 1972. Formal Aspects of Phonological Description. The Hague: Mouton.

[33] Kaplan, Ronald M. and Martin Kay. 1981. Phonological rules and finite-state transducers. In Linguistic Society of America Meeting Handbook, Fifty-Sixth Annual Meeting. New York. Abstract.

[34] Kaplan, Ronald M. and Martin Kay. 1994. Regular models of phonological rule systems. Computational Linguistics 20(3):331-378.

[35] Karttunen, Lauri. 1993. Finite state constraints. In John Goldsmith, ed., The last phonological Reflections on constraints and derivations, 173-194. University of Chicago Press.

[36] Karttunen, Lauri. 1995. The replace operator. In ACL'95. Cambridge, MA. cmplg/ 9504032.

[37] Karttunen, Lauri, Ronald M. Kaplan, and Annie Zaenen. 1992. Two-level morphology with composition. In COLING'92, pages 141-148. Nantes, France.

[38] Kenstowicz, Michael, and Charles Kisseberth. 1979. Generative phonology. Academic Press, Inc.

[39] Kirtchuk, Pablo 1987. Le parler quechua de Santiago del Estero (Argentine): quelques particularités. Amerindia 12: 95- 110. París.

[40] Klein, Harriet Manelis. 1978. Una gramática de la lengua toba: morfología verbal y nominal. Montevideo: Universidad de la República (ed. en inglés 1974).

[41] Koskenniemi K. 1979. Two-level Morphology: A general Computational Model for Word- Form Recognition and Production. Ph.D. thesis, University of Helsinki. Publications nº 11.

[42] Koskenniemi K. and Church K.W. 1988. Complexity, two-level morphology and Finnish. Proc. of 12th COLING, 335-340.

[43] Krivoshein de Canese, Natalia. 1994. Gramática de la lengua guaraní. Asunción: Colección Nemity.

[44] Messineo, Cristina. 2003. Lengua toba (guaycurú). Aspectos gramaticales y discursivos. LINCOM Studies in Native American Linguistics 48. Muenchen: LINCOM EUROPA Academic Publisher.

[45] Ritchie G. 1992. Languages generated by two-level morphological rules, Computational Linguistics, vol 18, No.1.

[46] Rosenberg, A.L. 1967. A Machine Realization of the linear Context-Free Languages, Information and Control 10, 177-188.

[47] Schane, Sanford A. 1973. Generative phonology. Englewood Cliffs, NJ: Prentice-Hall, Inc.

[48] Schane, Sanford A. 1979. Introducción a la fonología generativa. Barcelona, Ed. Labor.

[49] Schützenberger, Marcel-Paul. 1961. A remark on finite transducers. Information and Control 4:185-196.

[50] Seki, Lucy. 2000. Gramática do Kamaiurá, Língua Tupi-Guarani do Alto Xingu. Editora UNICAMP and São Paulo State Official Press.

[51] Sproat, R. 1992.Morphology and Computation. The MIT Press.

[52] Stark, Louisa R. 1985. History of the Quichua of Santiago del Estero. En: Klein y Stark (eds.), South American IndianLanguages: 732-752. Austin: University of Texas Press.

[53] Sudkamp, T.A.. 2006. Languages and Machines. An Introduction to the Theory of Computer Science, Pearson Addison-Wesley, 3ra. edición.

[54] Viñas Urquiza, M.T. 1974. Lengua mataca 1 y 2. Buenos Aires. UBA.

[55] Leo Wetzels (ed.). 1995. Estudos Fonológicos de Língüas Indigenas Brasileiras. Editora da UFRJ. Rio de Janeiro.

[56] Younger, D. H. 1967. Recognition and parsing of context-free languages in time n3. Information and Control, 10:189-208.