Capítulo 4

Modelización de símbolos texturados

Este capítulo presenta una revisión de diferentes tipos de gramáticas así como su proceso de inferencia, y de análisis sintáctico o parser, siempre desde el punto de vista del reconocimiento sintáctico de patrones. A continuación, muestra el modelo de símbolo texturado definido en este trabajo para la representación de símbolos texturados, así como su proceso de inferencia. Este modelo es una gramática de grafo a la que le añadimos producciones de error para poder tolerar ciertas distorsiones en los símbolos a reconocer. Para terminar presentamos algunos resultados de la inferencia gramatical y el proceso de parser.

4.1 Introducción

Un símbolo texturado es, como hemos visto en el Capítulo 1, un patrón con una estructura definida, en parte, por una o varias texturas estructuradas. Las aproximaciones sintácticas y estructurales, que realizan este tipo de reconocimiento, se basan en el uso explícito de la estructura que define el patrón. Una representación estructural de un patrón consiste en definir cómo un patrón complejo se construye de forma recursiva a partir de componentes más simples. Esta representación incluye, no sólo qué partes simples forman el patrón, sino qué relaciones existen entre ellas. Dentro del enfoque sintáctico, las gramáticas formales se usan para representar una clase de patrones. Las producciones de la gramática describen cómo los patrones se construyen a partir de partes simples. El proceso de reconocimiento se basa en el concepto de analizador sintáctico (o parser) de una gramática formal.

En este capítulo vamos a definir que es una gramática formal y los diferentes tipos de gramáticas que podemos encontrar. A continuación explicaremos los diferentes tipos de analizadores sintácticos que podemos encontrar. Para pasar a definir la inferencia gramatical, es decir cómo podemos deducir de forma automática una gramática a partir de un conjunto de patrones que han de ser definidos por ésta. Al final explicaremos cómo hemos resuelto el problema de representar y reconocer símbolos texturados usando gramáticas.

4.2 Gramáticas

Una gramática es un conjunto de reglas que definen la sintaxis de un lenguaje o grupo de patrones. Aunque se han usado en diversos ámbitos nosotros nos centraremos en el ámbito del reconocimiento sintáctico de patrones. Dentro de este ámbito se han utilizado en el subcampo del reconocimiento de caracteres, reconocimiento de notación musical, reconocimiento de fórmulas matemáticas, análisis de documentos, interpretación de dibujos lineales, análisis de ECG, reconocimiento de voz, reconocimiento de objetos 3-D y análisis de secuencias de DNA.

Una definición más precisa de gramática en el campo concreto del reconocimiento de patrones es, un conjunto de reglas que describen de forma recursiva cómo se pueden construir o generar los patrones de una o varias clases a partir de subpatrones simples. Para poder definir formalmente una gramática necesitamos definir primero algunos conceptos:

Definición 4.1 Un *alfabeto* A es un conjunto de símbolos. Una palabra x sobre A es una secuencia de símbolos $x = a_1 \dots a_n$ donde $a_i \in A$; $i = 1, \dots, n$. La palabra vacía ϵ es una secuencia sin símbolos.

Definición 4.2 La *longitud* de una palabra x se denota por $|x|$, y es igual al número de símbolos que la forman. Así tenemos que $|\epsilon| = 0$

Definición 4.3 El conjunto de todas las palabras sobre un alfabeto A se denota por A^*.

Definición 4.4 El conjunto de todas las palabras sobre un alfabeto A excepto la palabra vacía se denota por A^+, es decir $A^+ = A^* - \{\epsilon\}$.

Definición 4.5 Denotamos por a^n la *concatenación* de n ocurrencias del símbolo a. La concatenación de dos palabras $x = a_1 \dots a_n$ y $w = b_1 \dots b_m$ se denota por $xw = a_1 \dots a_n b_1 \dots b_m$. Hemos de observar que $x\epsilon = \epsilon x = x$, y que $(xw)z = x(wz)$ para cualquier $x, w, x \in A^+$. Esta concatenación puede extenderse a conjuntos de palabras X e W definiéndose $XW = \{x, w | x \in X, w \in W\}$.

Definición 4.6 Una *gramática formal* es una 4-tupla $G = (N, T, P, S)$ donde:
N es un conjunto finito de símbolos no terminales,
T es un conjunto finito de símbolos terminales,
P es un conjunto finito de producciones o reglas de reescritura,
$S \in N$ es el símbolo inicial,
$V = N \bigcup T$ es el vocabulario.
Y se cumple que:
$N \bigcap T = \emptyset$,
Cada producción $p \in P$ es de la forma $\alpha \to \beta$, donde α y β se llaman lado izquierdo y lado derecho de la regla respectivamente, con $\alpha \in V^+$ y $\beta \in V^*$.

La parte más importante de una gramática es su conjunto de producciones. Dada una gramática G y una producción en P $\alpha \to \beta$, decimos que cualquier aparición de α en una palabra puede ser reemplazada por β. Para definirlo de manera formal:

Definición 4.7 Dada una gramática G y una producción $\alpha \to \beta$ en P. Cualquier palabra $v = x\alpha y$ con $x, y \in V^*$ se puede *derivar a* una palabra $w = x\beta y$ en un paso, y lo denotamos como $v \to w$. Si en vez de derivar en un paso tenemos que existen las palabras $v = v_0, v_1, \ldots, v_n = w$, tales que $v_i \to v_{i+1}; i = 0, \ldots, n-1$, entonces decimos que v deriva a w en cero o más pasos, y lo denotamos por $v \overset{*}{\to} w$.

Definición 4.8 El *lenguaje generado por una gramática* G se define como:

$$L(G) = \{x | x \in T^*, S \overset{*}{\to} x\}$$

Así en la derivación de un elemento $x \in L(G)$ se empieza por el símbolo inicial S y se aplican reglas sucesivamente hasta que se obtiene una palabra w que contiene sólo símbolos del alfabeto terminal.

Las gramáticas podrían clasificarse en función del tipo de sus producciones, o en función del tipo de las estructuras que generan.

Jerarquía de Chomsky:

- **Sin restricciones, o de tipo 0:** Este tipo de gramáticas no tienen ningún tipo de restricciones.
- **Sensibles al contexto, o de tipo 1:** Sus producciones son de la forma $xAy \to xzy$, donde $x, y \in V^*$; $A \in N$ y $z \in V^+$. Es decir, el símbolo no terminal A puede ser reemplazado por la cadena no vacía z sólo si tiene como contexto derecho x y como izquierdo y.
- **Libres de contexto, o de tipo 2:** Todas las producciones son de la forma $A \to z$, donde $A \in N$ y $z \in V^+$. Cualquier regla en una gramática libre de contexto describe el reemplazo de un único símbolo no terminal A por una palabra no vacía z, por ello el reemplazo es independiente del contexto de A.
- **Regulares, o de tipo 3:** Todas sus producciones son del tipo $A \to aB$ o $A \to a$, donde $A, B \in N$ y $a \in T$. Estas gramáticas son un tipo especial de gramáticas libres de contexto donde la parte derecha de la producción cumple restricciones adicionales.

Clasificación de gramática según el tipo de estructura que genera:

- **Gramáticas lineales:**
  - **Gramáticas de cadena:** Generan cadenas.
  - **Extensión de las gramáticas de cadena:** Son extensiones de las gramáticas de cadena que, mediante la introducción de determinados operadores o reglas de conexión, permiten representar formas bidimensionales. En cierto modo pueden considerarse también casos particulares de las gramáticas de grafo.
    - PDL

  - Gramáticas plex.

- **Gramáticas n-dimensionales:**
  - **Gramáticas de vector:** Generan vectores de n dimensiones.
  - **Gramáticas de árbol:** Generan árboles.
  - **Gramáticas de grafo:** Generan grafos.

Definición 4.9 Un lenguaje se dirá que es de tipo i si es generado por una gramática de tipo i

Definición 4.10 Dada una gramática libre de contexto $G = (N, T, P, S)$, un *árbol de derivación* es un árbol donde:
(1) Cada nodo se etiqueta con un símbolo $z \in V$ tal que:
- Cada nodo hoja se etiqueta con un símbolo $a \in T$.
- Cada nodo intermedio se etiqueta con un símbolo $A \in N$.
- La raíz se etiqueta con el símbolo inicial S.

(2) Si existe un nodo con una etiqueta $A \in N$ tal que sus nodos sucesores se etiquetan con $x_1, \ldots, x_n \in V$ entonces existe una producción $A \to x_1, \ldots, x_n$ en P.

Definición 4.11 Dada una gramática libre de contexto $G = (N, T, P, S)$. Una palabra $x \in L(G)$ se llama *no ambigua* si existe un solo árbol de derivación de x respecto a G. La palabra se llamará *ambigua* si existe más de un árbol de derivación.

Una gramática formal es una herramienta que permite describir un conjunto infinito de cadenas, es decir un lenguaje $L(G)$. En el campo del reconocimiento de patrones, las gramáticas formales están formadas por producciones cuyos símbolos terminales corresponden a primitivas o componentes elementales de un patrón, que pueden ser extraídos directamente de un patrón de entrada o por medio de un método de preprocesado o segmentación, mientras que los no terminales son subpatrones con una complejidad mayor. Estos subpatrones se construyen de forma jerárquica a partir de los elementos primitivos. El proceso de construcción de subpatrones complejos a partir de partes simples se modela mediante producciones de la gramática. El lenguaje generado por la gramática representa a la clase completa del patrón.

Dadas N clases de patrones, el problema de clasificar un patrón desconocido se puede resolver construyendo una gramática G_i para cada clase de patrones, y analizando sintácticamente x de acuerdo con cada G_i para $i = 1, \ldots, N$. Una ilustración gráfica de esta explicación se presenta en la Fig. 4.1. Si $x \in L(G_i)$ se decide que x pertenece a la clase de patrones i. Si no existe una G_i tal que $x \in G_i$ entonces x será rechazado. Debemos observar que como subproducto del análisis sintáctico obtenemos un árbol sintáctico de x de acuerdo con G_i siempre que $x \in G_i$. Esto nos proporciona información sobre qué subpatrones simples componen x. Es decir el análisis sintáctico no sólo sirve para clasificar patrones sino para obtener la interpretación jerárquica de los símbolos que componen un patrón de entrada en función de una gramática subyacente.

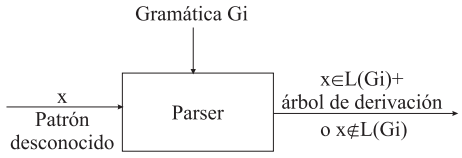


Figura 4.1: Clasificación de símbolos mediante gramáticas.

Hasta ahora hemos hablado principalmente de gramáticas de cadena. En las próximas subsecciones vamos a presentar con mayor detalle otros tipos de gramáticas. Primero veremos extensiones de las gramáticas de cadena que permiten un mayor número de dimensiones, como son las PDL y las *plex*, para seguir con gramáticas n-dimensionales por naturaleza como son las de vector, grafo y árbol. Terminaremos el apartado con dos generalizaciones de gramáticas que pueden ser aplicadas a todas las mencionadas anteriormente incluidas las de cadena. Estas generalizaciones son las gramáticas de atributos y las programables.

4.2.1 Extensión de las gramáticas de cadena, en mayor número de dimensiones

Las gramáticas de cadena son muy restrictivas cuando quieren describir patrones de dos o más dimensiones. Existen varias extensiones para tener gramáticas que permitan representaciones más efectivas de patrones con mayor número de dimensiones.

Lenguaje de descripción pictórica (PDL)

Una de estas extensiones es el lenguaje de descripción pictórica, en inglés *picture description language*, (PDL). En este lenguaje cada primitiva de un patrón tiene exactamente dos puntos llamados cabeza y cola, por donde se puede conectar a otras primitivas. Formalmente cada primitiva se puede representar como una arista dirigida y etiquetada de un grafo. En PDL hay cuatro formas de conectar un par de primitivas. Estas cuatro formas están representadas por cuatro operaciones binarias que se denotan como $+$, $-$, $\times$ y $*$. Además hay un operador unario, $\tilde{}$ que significa intercambiar la cola y la cabeza de una primitiva. El funcionamiento de los operadores binarios así como su interpretación geométrica pueden verse reflejados en la Tabla 4.1. Cada entidad resultado de encadenar dos primitivas tiene a su vez una cabeza, *ca* y una cola, *co* como se muestra en la Tabla 4.1. Así se pueden extender los cuatro operadores binarios a estructuras de nivel superior, es decir, podemos usar estructuras más complejas de PDL como argumentos de operadores. Se puede ver que cualquier grafo dirigido con aristas etiquetadas puede ser representado por una expresión PDL. Consecuentemente existe una expresión PDL para cada patrón que se pueda representar por un grafo de elementos primitivos.

En la aplicación de los PDL al reconocimiento de patrones, una clase completa de patrones se representa por una gramática que genera un conjunto de expresiones PDL. El reconocimiento de un patrón desconocido se lleva a cabo convirtiendo éste en su representación simbólica usando PDL y luego analizándolo sintácticamente de

acuerdo con una gramática subyacente. Por ejemplo, para representar el patrón que aparece en la Fig. 4.2(a), podemos crear una gramática a partir de las primitivas de la Fig. 4.2(b), y así el patrón quedaría codificado como se presenta en la Fig. 4.2(c), esta gramática se define como sigue:

$G = (N, T, P, S)$, donde:

- $N = \{S\}$
- $T = \{a, b, +, -, \times, *, \sim, (,)\}$
- $P = \{S \rightarrow (b + a + (\ b)) \times (a)\}$
- $L(G) = \{(b + a + (\sim b)) \times (a)\}$

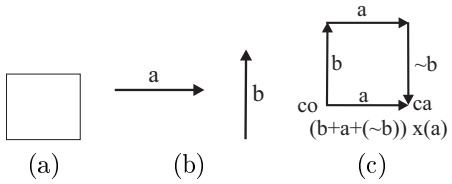


Figura 4.2: Codificación de un patrón mediante una gramática PDL: (a)Patrón. (b)Primitivas a partir de las que generamos el patrón. (c) Patrón codificado usando PDL.

Además de patrones con una forma fija podemos definir una gramática que represente un símbolo texturado como el presentado en la Fig. 4.3(a). Esta gramática debe permitir la generación de una cuadrícula de cualquier tamaño, y es como sigue:

$G = (N, T, P, S)$, donde:

- $N = \{S, Linea, Cuad, RCuadDer, CuadDer, RLineaIzq, LineaIzq, CuadInfIzq, RLineaDer, LineaDer, CuadInfDer\}$
- $T = \{a, b, +, -, \times, *, \sim, (,)\}$
- $P =$

$$
\begin{array}{l}
\{ \\
S \rightarrow Linea \ \mid Linea + RLineaIzq, \\
Linea \rightarrow Cuad + RCuadDer, \\
Cuad \rightarrow (b + a + (\sim b)) \times (a), \\
RCuadDer \rightarrow CuadDer \mid CuadDer + RCuadDer, \\
CuadDer \rightarrow a - (a + (\sim b)), \\
RLineaIzq \rightarrow (\sim b) + LineaIzq \mid (\sim b) + LineaIzq + RLineaDer, \\
LineaIzq \rightarrow CuadInfIzq \mid CuadInfIzq + LineaIzq, \\
CuadInfIzq \rightarrow ((\sim a) - (\sim b)), \\
RLineaDer \rightarrow (\sim b) + LineaDer \mid (\sim b) + LineaDer + RLineaDer, \\
LineaDer \rightarrow CuadInfDer \mid CuadInfDer + LineaDer, \\
CuadInfDer \rightarrow (a - (\sim b)), \\
\}
\end{array}
$$

- $L(G) = \{((b + a + (\sim b)) \times a)$,
  $(((b + a + (\sim b)) \times a) + (a - (a + (\sim b))))$,
  $(((b + a + (\sim b)) \times a) + (a - (a + (\sim b)))) + (a - (a + (\sim b))))$,
  $(((b + a + (\sim b)) \times a) + (a - (a + (\sim b)))) + (a - (a + (\sim b)))) + (a - (a + (\sim b))))$,
  $(((b + a + (\sim b)) \times a) + (a - (a + (\sim b)))) + (a - (a + (\sim b)))) + (a - (a + (\sim b))))$,
  $(((b + a + (\sim b)) \times a) + (a - (a + (\sim b)))) + (a - (a + (\sim b)))) + (a - (a + (\sim b))))$,
  $\ldots$,
  $(((b + a + (\sim b)) \times a) + (a - (a + (\sim b))) + (a - (a + (\sim b))) + (a - (a + (\sim b))) + (\sim b) + ((\sim a) - (\sim b)) + ((\sim a) - (\sim b)) + ((\sim a) - (\sim b)) + (\sim b) + (a - (\sim b)) + (a - (\sim b)) + (a - (\sim b)) + (a - (\sim b)))$,
  $\ldots\}$

Podemos ver que el lenguaje, $L(G)$, generado por esta gramática es infinito, aquí hemos presentado algunas de las palabras que generaría, entre ellas está la última que corresponde a la cuadrícula de la Fig. 4.3(a), presentada en su forma PDL en la Fig. 4.3(b). El problema de este tipo de gramáticas es que para generar esta cuadrícula también estamos permitiendo la generación de otros patrones como los presentados en la Fig. 4.4, además para configuraciones más complicadas la generación de la gramática también se complica.

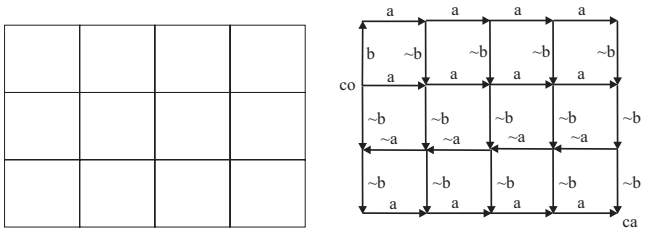


Figura 4.3: (a) Símbolo texturado. (b)Representación simbólica del símbolo texturado usando PDL.

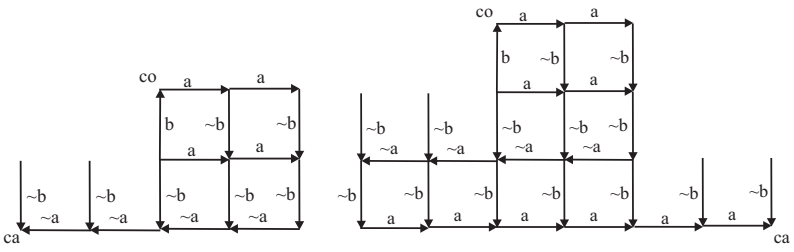
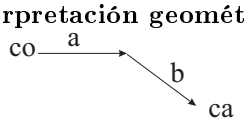
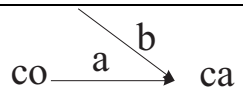
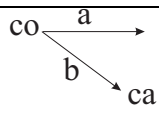
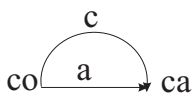


Figura 4.4: Palabras del lenguaje generadas por una gramática PDL que genera cuadrículas que no se corresponden a la textura de cuadrícula esperada.

Tabla 4.1: Operadores binarios de PDL.

Operador	Significado	Interpretación geométrica
a+b	cabeza(a) conectada con cola(b) cabeza(a+b)=cabeza(b) cola(a+b)=cola(a)	
a-b	cabeza(a) conectada con cabeza(b) cabeza(a-b)=cabeza(b) cola(a-b)=cola(a)	
axb	cola(a) conectada con cola(b) cabeza(axb)=cabeza(b) cola(axb)=cola(a)	
a*c	cola(a) conectada con cola(c) y cabeza(a) conectada cabeza(c) cabeza(a*c)=cabeza(c) cola(a*c)=cola(a)	

Gramáticas plex

El problema del PDL es su limitación a dos puntos de concatenación de sus primitivas. Para arreglar este problema se crearon las estructuras *plex*. Este tipo de estructuras consiste en una lista de primitivas, una lista de conexiones internas y otra de conexiones externas. A diferencia de las estructuras con PDL, los elementos primitivos en una estructura plex pueden tener cualquier número de puntos de conexión. Una gramática plex consiste en no terminales, terminales, un símbolo de inicio y un conjunto de producciones. Un lenguaje generado por una gramática plex son todas las estructuras plex terminales que se pueden generar a partir del símbolo inicial por medio de las producciones.

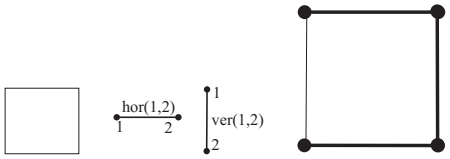
Las gramáticas plex se utilizan en el campo del reconocimiento de patrones de una forma muy similar a las gramáticas PDL. Primero una clase de patrones se representa por medio de una gramática plex. Después, dado un patrón a reconocer, primero se convierte a su representación simbólica en términos de su estructura plex, y luego se analiza sintácticamente si cumple las reglas de la gramática.

En la aplicación de las plex al reconocimiento de patrones, una clase completa de patrones se representa por una gramática que genera un conjunto de expresiones plex. El reconocimiento de un patrón desconocido se lleva a cabo convirtiendo éste en su representación simbólica usando plex y luego analizándolo sintácticamente de acuerdo con una gramática subyacente. Por ejemplo, para representar el patrón que aparece en la Fig. 4.5(a), podemos crear una gramática a partir de las primitivas de la Fig. 4.5(b), y así el patrón quedaría codificado como se presenta en la Fig. 4.5(c), esta gramática se define como sigue:

$G = (N, T, P, S)$, donde:

- $N = \{< CUAD >\}$

- $T = \{ver(1,2), hor(1,2)\}$
- $S = CUAD$
- $P = \{CUAD \rightarrow hor\ ver\ hor\ ver(2100, 0220, 0012, 1001)\}$



(a) (b) (c)

Figura 4.5: Codificación de un patrón mediante una gramática plex: (a)Patrón. (b)Primitivas a partir de las que generamos el patrón. (c) Patrón codificado usando Plex: (ver,hor,ver,hor)(2100,0220,0012,1001)().

4.2.2 Gramáticas de vector

Las cadenas pueden considerarse como vectores de una dimensión así que podríamos extender este concepto a gramáticas de vector.

Gramáticas isométricas

Para poder extender el concepto de gramática de cadena a gramática de matriz necesitamos modificar la definición de gramática.

Definición 4.12 Una *gramática isométrica* de cadena es una 5-tupla $G = (N, T, P, S, \#)$, donde N, T, P y S corresponden a la definición general de gramática de cadena, mientras que $\# \in N$ se llama el símbolo en blanco. Además es necesario que las reglas $\alpha \rightarrow \beta$ en P cumplan las siguientes condiciones:
(a) $|\alpha| = |\beta|$
(b) α no consiste sólo en #.
(c) Reemplazar α por β no debe suponer desconectar o eliminar los símbolos diferentes a #.
en realidad todas estas reglas significan que los símbolos que no son # en β existen y están conectados, y si α tiene un símbolo no # como final izquierdo o derecho entonces β también lo tiene.

Definición 4.13 El *lenguaje* $L(G)$, siendo G una gramática isométrica, es el conjunto de cadenas de terminales x, tal que la cadena infinita $\#^\infty x\#^\infty$ se puede derivar de G a partir de la cadena infinita inicial $\#^\infty S\#^\infty$.

Gramáticas paralelas

En una gramática ordinaria, cuando estamos en un paso cualquiera de la derivación, una regla dada puede aplicarse en varios puntos pero sólo la aplicamos en uno. En una gramática paralela las reglas se aplican en todos los puntos posibles y de forma simultanea.

Gramáticas de matriz

La clase de gramáticas que definen un vector rectangular puede definirse como: Una gramática que genera una cadena α que será la primera fila de la matriz. Los símbolos que forman la cadena α serán símbolos iniciales de un conjunto de gramáticas de estados finitos $G_1, \ldots, G_n$. Una vez se ha generado α, estas gramáticas operan en paralelo para generar las columnas de la matriz. La aplicación de las reglas en paralelo debe ser de tal forma que se asegure que en todo momento y en cada columna se aplica una regla de la misma longitud, y que las reglas de terminación se aplican al mismo tiempo. Para definirlas de una manera formal:

Definición 4.14 Una *gramática de matriz* G es una tupla de $(n+1)$ componentes $G', G_1, \ldots, G_n$, donde G' es una gramática y $G_1, \ldots, G_n$ son gramáticas de estados finitos tales que el vocabulario terminal de G' es el conjunto $S_1, \ldots, S_n$ de símbolos iniciales de las gramáticas G_i, $i = 1, \ldots, n$. G opera generando la cadena α de S_i, $i = 1, \ldots, n$ usando las reglas de G', para posteriormente generar una matriz rectangular desde la fila inicial α aplicando las reglas de las gramáticas G_i en paralelo, teniendo el conjunto de reglas aplicado en cada paso la misma longitud, y o todas son no terminales o todas son terminales.

Gramáticas de vector n-dimensional

Son gramáticas que generan vectores de símbolos conectados entre sí. Un vector de este tipo se puede ver como la correspondencia de un conjunto de puntos (i, j) de un entramado con un vocabulario V, donde todos excepto algunos puntos de enrejado se corresponden con el símbolo blanco #. Es decir, un vector se construye colocando los símbolos del vocabulario V en coordenadas enteras del plano (i, j). Dos símbolos se llaman vecinos si están en posiciones (i, j),(k, l) tales que $|i - k| + |j - l| = 1$. Un vector W se llama conectado si para todos los símbolos no blancos $(no - \#)$, a, b en W, existe una secuencia $a = a_0, a_1, \ldots, a_n = b$ de símbolos $no - \#$ en W tales que a_i es vecino de a_{i-1}, $1 \leq i \leq n$.

Una gramática de vector n-dimensional se define igual que las gramáticas isométricas de cadena, solo que las reglas son pares de vectores conectados en vez de pares de cadenas. La razón por la que necesitamos que las gramáticas de vector sean isométricas es porque la derivación actúa reemplazando subvectores por subvectores, igual que las gramáticas de cadena actúan reemplazando subcadenas por subcadenas. Sin embargo, si dos subvectores α y β no son idénticos en forma y medida no está claro como reemplazar α por β. Se podría hacer moviendo filas o columnas en el vector original para dar cabida a β, pero eso cambiaría, de forma arbitraria, las vecindades de posiciones lejanas a β, dado que los movimientos de filas y columnas serían de

diferente tamaño. Así cuando se necesite una gramática de vector isométrica, para cualquier regla $\alpha \rightarrow \beta$, α y β serán geométricamente idénticas, y por tanto será obvio como reemplazar α por β. Como en el caso de las cadenas, la manera de hacer crecer el vector será añadiendo el símbolo #.

Definición 4.15 Una *gramática de vector* isométrica es una 5 tupla $G = (N, T, P, S, \#)$, donde N, T, #, y S se definen como en la definición de gramática isométrica general y P es un conjunto de pares de vectores conectados (α, β) para los que:

(a) α y β son geométricamente idénticos.
(b) α no consiste sólo en #.
(c) β satisface las condiciones:
  (c.1) Si los símbolos $no-\#$ de α no tocan la frontera de α, entonces los símbolos $no-\#$ de β deben estar conectados (y no ser vacíos).
  (c.2) Sino,
    (c.2.1) Cada componente conectado de símbolos $no-\#$ en β debe contener la intersección de algún componente de símbolos $no-\#$ en α con la frontera de α.
    (c.2.2) A la inversa, cada una de esas intersecciones debe contener en algún componente un $no-\#$ en β.

El lenguaje definido por G, $L(G)$, es el conjunto de todos los vectores W con terminales conectados no en blanco. Tal vector W, empotrado en un vector con infinitos #) puede ser derivado en G a partir de un vector inicial, que consiste en un único S empotrado en un vector con infinitos #. Las derivaciones se realizan como en el caso de las gramáticas de cadena, solo que en este caso en vez de reemplazar unas subcadenas por otras, reemplazamos unos subvectores por otros.

4.2.3 Gramáticas de grafo

Tanto las gramáticas PDL como las plex generalizan las gramáticas de string añadiendo símbolos especiales de unión. Pero existe una forma universal de representar datos de dos o más dimensiones que son los grafos etiquetados. Si se usan grafos en lugar de cadenas como estructuras de datos fundamentales de la gramática entonces tenemos *gramáticas de grafo*. Conceptualmente una gramática de grafo es muy similar a una gramática de cadena consiste en un conjunto de etiquetas no terminales para nodos y para aristas, un conjunto de etiquetas terminales para nodos y aristas, un grafo inicial y un conjunto de producciones. La principal diferencia con las gramáticas de cadena es que las producciones en las gramáticas de grafo están formadas por una parte derecha y una parte izquierda que son grafos. En el caso concreto de las gramáticas de grafo libres de contexto la parte izquierda es un único nodo. El lenguaje generado por una gramática de este tipo son todos los grafos con etiquetas terminales en sus nodos y aristas que pueden generarse por medio de las reglas de la gramática a partir del grafo inicial. Aplicar una producción de la gramática consiste en reemplazar una ocurrencia de la parte izquierda de la producción por la parte derecha de la misma. Normalmente la parte izquierda de la producción, que tiene que ser reemplazada, la encontramos inmersa en función de un número de aristas que la conectan a un grafo a analizar. Una vez reemplazamos esa parte izquierda por la parte derecha de la

producción esa inmersión se pierde. Para poder especificar la forma de conectar el subgrafo derecho en el grafo restante después de eliminar el subgrafo izquierdo de la producción se han de definir unas reglas de inmersión. Así las producciones necesitan tener además de la parte izquierda y la parte derecha, unas reglas de inmersión que definan como quedará conectado el grafo derecho cuando sustituya al izquierdo en el grafo restante.

Cadenas y vectores pueden considerarse un caso especial de grafos etiquetados. Podemos ver los símbolos de una cadena como etiquetas asociadas a los nodos de un grafo, donde cada nodo tiene dos vecinos, excepto los correspondientes a los extremos de la cadena que tienen sólo uno. De forma similar podemos ver una matriz como un grafo, cuyos nodos están etiquetados con los símbolos de la matriz y en el que cada nodo tiene cuatro vecinos como máximo.

Cuando las gramáticas de grafos etiquetados fueron introducidas por primera vez estos grafos etiquetados se llamaron *webs*. La manera más simple de definir una gramática de grafo o web es la siguiente:

Definición 4.16 Una *gramática de grafo o web* es una 6 tupla $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$, donde:
Σ es el alfabeto de las etiquetas no terminales de los nodos.
Δ es el alfabeto de las etiquetas terminales de los nodos.
Γ es el alfabeto de las etiquetas no terminales de las aristas.
Ω es el alfabeto de las etiquetas terminales de las aristas.
P es el conjunto finito de producciones de la gramática o reglas de reescritura.
S es el conjunto de grafos iniciales, que normalmente consisten en un nodo con una etiqueta no terminal.
Una producción de una gramática G es una 3 tupla $P = (l, r, T)$, donde:
l es el grafo de la parte izquierda de la regla.
r es el grafo de la parte derecha de la regla.
T es la transformación de inmersión.

Dado un grafo H que tenga un subgrafo isomórfico con el subgrafo l, denotaremos al subgrafo de H isomofico al l, como l^H y al grafo H sin l^H como H^{-l^H}.

De forma similar se pueden definir gramáticas que generen grafos dirigidos, llamados "digrafos" para abreviar. En este tipo de gramáticas la parte derecha e izquierda de una regla serán dos grafos dirigidos, r y l, y las reglas de empotrado deben especificar la dirección de las aristas con las que conectan el subgrafo r en el grafo restante H_{-l^H}, a un grafo H dado.

Así en una derivación aplicamos la regla o producción $P = (l, r, T)$ a un grafo H reemplazando una ocurrencia del subgrafo l en H, l^H, por r. Sin embargo aquí aparece un problema que no existe en las gramáticas de cadena ni en las de vectores porque necesitamos que sean isométricos. El problema es cómo hacer la inmersión del subgrafo isomórfico con r en H^{-l^H} en la posición de l. Para ello necesitamos que cada regla gramatical nos de esa información que llamaremos de "inmersión", es decir qué nodos de r serán vecinos, con arcos que los conecten, de qué nodos de H^{-l^H}. El lenguaje generado por una gramática de este tipo será el conjunto de grafos que tengan etiquetas terminales, y que puedan derivarse a partir de un nodo inicial etiquetado como S.

Según el tipo de inmersión que tengamos obtendremos unos resultados u otros. Existen tres tipos básicos de inmersión:

1. **De pegado:** En el que ciertas partes de H^{-l^H} se identifican con ciertas partes de r^H. Estas partes de H^{-l^H} se habían identificado previamente con partes de l.

2. **De conexión:** En el que se añaden nuevos arcos para conectar r^H con H^{-l^H}. Cuando l^H se borra de H también se borran todos los arcos que conectan nodos de l^H con nodos de H que no pertenecen a l^H.

3. **Lógicas:** Basan sus reglas de inmersión en predicados lógicos.

En función del tipo de inmersión que escojamos tendremos un tipo de gramática de grafo:

1. **Aproximación algebraica:** Se basa en la inmersión de pegado. Se llama así porque se basa en una construcción algebraica. Sus reglas son menos expresivas en cuanto a los arcos de los nodos borrados, pero más adecuadas para la definición de transformaciones de grafos en paralelo.

2. **Aproximación algorítmica (o de teoría de conjuntos):** Se basa en la inmersión de conexión. Se llama algorítmica debido a que sus definiciones son de tipo constructivo. Dentro de este tipo existen diversos subtipos en función de: El tipo de estructura subyacente, básicamente si son digrafos o hipergrafos; La forma de la parte izquierda de las reglas, si es un solo nodo o un subgrafo; La forma de copiar la parte derecha de la regla en el resto del grafo.

3. **Aproximación lógica:** Se basa en las reglas de inmersión basadas en la lógica. Básicamente existen de dos tipos: Una basada en predicados monádicos de segundo orden y otra basada en predicados de primer orden.

El problema de las gramáticas de grafo es la dificultad y elevada complejidad computacional de sus analizadores sintácticos. Por eso existen otros tipos de gramáticas que son simplificaciones de las de grafo, como la de árbol o la de array. Sin embargo, debido al gran poder de representación que tienen, y a la posibilidad de reducir su complejidad en casos reales mediante restricciones añadidas gracias al conocimiento previo de las características del entorno en que se van a utilizar, se han utilizado en diversos campos como: el reconocimiento de diagramas [17], la notación musical [39], lenguajes visuales [84], reconocimiento de cotas [32, 75] y reconocimiento de fórmulas matemáticas [53].

4.2.4 Gramáticas de árbol

Como una cadena es un grafo acíclico y dirigido, es decir un árbol dirigido, podemos también considerar gramáticas de árbol. De hecho un caso especial de gramática de grafo sería una gramática de árbol. Cada regla $\alpha \rightarrow \beta$ tendría como α y β dos subárboles, y se aplicaría sobre un árbol H en el que sustituiríamos una ocurrencia del subárbol α en H por el subárbol β. En este caso el empotrado es fácil ya que

simplemente el único padre de la raíz de α será el padre de la raíz de β. El lenguaje generado por una gramática así será el conjunto de árboles etiquetados con etiquetas terminales en sus nodos que se puedan generar a partir de un árbol con un sólo nodo etiquetado con la etiqueta inicial S.

Una ventaja importante de las gramáticas de árbol respecto a las de grafo es que las primeras generan árboles haciendo que éstos crezcan a partir de sus hojas finales por medio de reescrituras de subárboles completos, y esta característica hace que el análisis sintáctico sea más fácil.

4.2.5 Generalizaciones de gramáticas

Gramática de atributos, o parametrizadas

A pesar de que las reglas gramaticales son una buena herramienta para modelizar de forma estructural las propiedades de los patrones, describiendo su descomposición jerárquica en componentes más simples, tienen deficiencias a la hora de representar información cuantitativa; como la orientación o longitud de líneas o parámetros de textura de las regiones, o la orientación 3D de superficies. Una solución a este problema son las gramáticas de atributos. La idea es aumentar cada símbolo $Y \in V$ de la gramática con un vector de valores de atributos $m(Y) = (x_1, \ldots, x_k)$, donde cada atributo α es una función $\alpha : Y \to D_Y$ que pone en correspondencia un símbolo $Y \in V$ con el vector $x_1, \ldots, x_k \in D_Y$ de valores numéricos. Un vector de atributos se puede interpretar como un vector de características en el reconocimiento estadístico de patrones.

Si consideramos una producción libre de contexto $A \to B_1, \ldots, B_m$, donde $A \in N$, $B_i \in V$, existe normalmente un relación entre los atributos de los símbolos de la parte derecha y los de la izquierda. Así podemos distinguir dos casos. Primero que los atributos de la parte izquierda sean dependientes de los de la parte derecha, en este caso se llaman **atributos sintetizados**, es decir $\alpha(A) = f(\alpha(B_1, \ldots, \alpha, (B_m))$, o que los atributos de la parte derecha sean dependientes de los de la parte izquierda, en este caso son **atributos heredados**, es decir $\alpha(B_i) = g_i(\alpha(A))$, para $i = 1, \ldots, m$.

En el reconocimiento de abajo a arriba, se empieza con un patrón segmentado con todas las primitivas extraídas de manera que los atributos de los símbolos no terminales se pueden calcular de forma sucesiva durante el análisis, es decir los atributos serán sintetizados. Cuando realizamos un reconocimiento de arriba a abajo debemos conocer los atributos del símbolo inicial, y se irán heredando de forma sucesiva a los otros símbolos de la gramática.

Un caso especial de gramáticas de atributos son las **gramáticas de coordenadas**, en las cuales la coordenada de posición (tuplas de d números para d dimensiones) se asocian a cada símbolo. Se pueden modelar disposiciones espaciales de los símbolos usando este tipo de gramáticas en las que en cada regla $\alpha \to \beta$, α y β son tuplas de símbolos, es decir, $\alpha = (A_1, \ldots, A_n)$, $\beta = (B_1, \ldots, B_m)$, y asociamos a cada regla una dn tuplas de funciones, cada una de dm variables, que especifica las coordenadas de las $B's$ en términos de las coordenadas de las $A's$. El lenguaje es el conjunto de patrones de símbolos terminales derivables de esta forma a partir de un S llamado origen.

Gramáticas programables

Las gramáticas programables pueden aplicarse a gramáticas de cadena, vector o grafo, pero aquí las explicaremos para cadenas.

La diferencia básica entre una gramática programable y una convencional es que cada producción tiene asociada la producción que se aplicará a continuación. Así una gramática programable consiste en una gramática G, con sus reglas etiquetadas de $1, \ldots, n$ y un conjunto de n tripletas (i, U_i, V_i), $1 \leq i \leq n$, donde i es el número de la regla y U_i, V_i son conjuntos de números de reglas. El conjunto U_i representa el conjunto de reglas que permitimos que se apliquen si la regla i se ha podido aplicar con éxito, mientras que el conjunto V_i es el conjunto de reglas que permitimos que se apliquen si la regla i se ha intentado aplicar pero sin éxito. Inicialmente se puede aplicar cualquier regla del tipo $S \rightarrow \beta$. Una derivación se llama admisible si cumple las siguientes propiedades:

1. Empieza por una regla de la forma $S \rightarrow \beta$.
2. Sean las cadenas en tres pasos sucesivos de la derivación $\lambda \rightarrow \mu \rightarrow \nu$, donde μ se obtiene a partir de λ aplicando la regla i y ν se obtiene de μ aplicando la regla j. Entonces existe una secuencia de reglas $i_1, i_2, \ldots, i_m$,$m \geq 1$, tal que $i_1 \in U_i$,$i_2 \in V_{i_1}, i_3 \in V_{i_2}, \ldots, i_m \in V_{i_{m-1}}$

El lenguaje de una gramática programable es el conjunto de cadenas terminales que se obtiene a partir de S por medio de una derivación admisible.

4.3 Corrección de errores

Hasta este punto hemos hablado de reconocimiento de patrones sin distorsiones o ruido. Pero en la mayoría de casos reales estas distorsiones existen. Para estos casos se pueden aplicar tres estrategias diferentes. Primero, si tenemos un conocimiento completo y preciso de los posibles errores que nos podemos encontrar se puede ampliar la gramática ideal G, es decir sin distorsiones, con *producciones de error*. De esta manera obtendremos una gramática G' cuyo lenguaje $L(G')$ representará no solamente los patrones ideales sino los distorsionados. Segundo, si un patrón puede ser generado por más de una gramática, es decir hay diferentes clases de patrones que se solapan, de manera que $x \in L(G_i)$ y $x \in L(G_j)$ para $i \neq j$. En este caso se puede aplicar información de las frecuencias relativas, de patrones y subpatrones. Esta solución es un caso de *gramática probabilística*, en concreto una aproximación de gramática estocástica que veremos más adelante. Por último, si no se tiene conocimiento previo de las posibles distorsiones ni de las frecuencias relativas de patrones y subpatrones, podemos todavía establecer algunas asumciones generales implícitas sobre posibles errores y usar un algoritmo de análisis sintáctico con corrección de errores. La idea básica de una analizador de este tipo es, dada una gramática G y una palabra que $x \notin L(G)$, encontrar la cadena $y \in L(G)$ que se parezca más a x, y construir su árbol de derivación de acuerdo con esta gramática G.

4.3.1 Gramáticas con información probabilística

Gramáticas estocásticas

Existen aplicaciones en las que clases diferentes tienen partes en común. Es decir existen elementos x tales que pueden ser generados a partir de gramáticas diferentes. En estos casos la aplicación de gramáticas estocásticas puede ser de gran ayuda. Una gramática estocástica G se caracteriza por que cada elemento $x \in G$ tiene asociado un número $p(x|G)$ que nos dice la probabilidad de que x sea generado por G. Si se conoce la probabilidad a priori de G, $p(G)$ entonces se puede deducir a partir del teorema de Bayes que:

$$\begin{array}{l} si\, x \in L(G_i)\, y\, x \in L(G_j)\, entonces \\ x \in \left\{ \begin{array}{ll} clase\, i, & si\, p(G_i)p(x|G_i) > p(G_j)p(x|G_j) \\ clase\, j, & sino. \end{array} \right. \end{array} \tag{4.1}$$

La probabilidad $p(x|G)$, siendo x no ambigua, se obtiene analizando sintácticamente x de acuerdo con G, encontrando las producciones de G que se necesitan para generar x, y multiplicando sus probabilidades. Si x es ambigua, se calcula la probabilidad de x para cada derivación diferente y se suman todas esas probabilidades.

Definición 4.17

(a) Una *gramática estocástica* es una 4-tupla $G^S = (N, T, P^S, S)$ donde:
- N, T y S son el alfabeto no terminal, el terminal y el símbolo inicial, como en la definición básica de gramática.
- P^S es el conjunto finito de producciones de la forma $p_{ij} : A_i \rightarrow X_{ij}$ con $A_i \in N$; $X_{ij} \in V^+$; $0 < p_{ij} \leq 1$; $\sum_{j=1}^{ni} p_{ij} = 1$; $i = 1, \ldots, m$.

(b) La *gramática característica* $G = (N, T, P, S)$ de una gramática estocástica G^S se obtiene borrando los números p_{ij} de cada producción en P^S

Definición 4.18 Dada una gramática estocástica $G^S = (N, T, P^S, S)$. El *lenguaje* generado por ella se define como $L(G^S) = \{[x, p(x|G^S)] | x \in T^*, S \stackrel{*}{\rightarrow} x\}$.

Un problema en este tipo de gramáticas es que se debe conocer la probabilidad de cada producción. La inferencia de estas probabilidades puede ser un problema en un caso práctico.

Gramáticas con analizadores sintácticos con multiples selecciones .

En el apartado anterior hemos visto que la incertidumbre se presentaba en el problema de reconocimiento de patrones. Ahora veremos como tratar el problema cuando la incertidumbre está en discernir cuales son las primitivas que estamos encontrando. Este método es muy útil cuando debido a distorsiones o ruido, las primitivas no se pueden detectar de forma precisa. Así en vez de tener un patrón de entrada de la forma $x = x_{i1} \ldots x_{im}$, $x_{ij} \in T$, $T = \{x_1, \ldots, x_n\}$, donde cada símbolo terminal está determinado de forma unequívoca para cada posición, tendremos una secuencia de

vectores:

$$\begin{bmatrix} c_1(x_1) \\ c_1(x_2) \\ . \\ . \\ . \\ c_1(x_n) \end{bmatrix} \quad \begin{bmatrix} c_2(x_1) \\ c_2(x_2) \\ . \\ . \\ . \\ c_2(x_n) \end{bmatrix} \quad \ldots \quad \begin{bmatrix} c_m(x_1) \\ c_m(x_2) \\ . \\ . \\ . \\ c_m(x_n) \end{bmatrix} \tag{4.2}$$

donde $c_i(x_j)$ es una medida de la certeza de que el símbolo en la posición i, $i = 1, \ldots, m$, sea $x_j \in T$ $j = 1, \ldots, n$. Así tenemos múltiples posibilidades en cada posición con diferentes certidumbres. En función de la aplicación deberemos imponer la restricción $\sum_{j=1}^{n} c_i(x_j) = 1$; $i = 1, \ldots, m$. Dadas las certidumbres $c_1(x_{i1}), c_2(x_{i2}), \ldots, c_m(x_{im})$, definimos la certeza de la cadena $x_{i1}x_{i2}\ldots x_{im}$ como: $c(x_{i1}x_{i2}\ldots x_{im}) = \sum_{j=1}^{m} c_j(x_{ij})$.

En general una secuencia de vectores como el de la ecuación 4.3.1 corresponde a n^m cadenas diferentes cada una con su propia certeza. El analizador sintáctico deberá determinar que cadena x es compatible con la gramática y tiene la certeza máxima, es decir: $x \in L(G)$ y $c(x) = max\{c(y)|y \in L(G)\}$

4.3.2 Gramáticas con analizador sintáctico con corrección de errores

Dentro del análisis sintáctico de patrones encontramos que, a menudo, éstos presentan distorsiones. Una gramática, en general, espera una construcción sintáctica fija del patrón, pero si éste presenta distorsiones no será capaz de determinar que forma parte del lenguaje generado por ella pero con una distorsión. La idea de tener una gramática con analizador sintáctico con correcciones de error se basa en la idea de distancia de un patrón al lenguaje generado por una gramática. Así si tenemos dos gramáticas, G_1 y G_2 que representan dos tipos de patrones, y tenemos un patrón de entrada x distorsionado, es necesario determinar si x está más cerca de $L(G_1)$ o de $L(G_2)$. Para ello se ha de definir una función de distancia, $D(L(G_1), x)$, o de similitud, $S(L(G_1), x)$ entre el patrón, x, y el lenguaje generado por una gramática, $L(G_1)$, y un proceso que calcule esta distancia o similitud. Este proceso del cálculo de la función de distancia se llama reconocedor con corrección de error o parser con corrección de error. A partir del cálculo de $D(L(G_1), x)$ se pueden realizar dos procesos. El primero se llama reconocedor con corrección de error y consiste en encontrar la distancia de similitud entre un patrón dado y la palabra generada por la gramática que más se parece a él. El segundo es el parser con corrección de error y consiste en determinar qué palabra o palabras generadas por el lenguaje son las más cercanas al patrón de entrada, calcular su árbol sintáctico y determinar la distancia.

4.4 Análisis sintáctico. (Parser)

Una gramática puede utilizarse para generar o aceptar lenguajes. Así el lenguaje de una gramática será la estructura (cadena, árbol o grafo) que la gramática pueda

generar así como la estructura que pueda analizar sintácticamente, o aceptar. Normalmente analizar sintácticamente una estructura es más difícil que generarla.

Un autómata, o máquina de Turing, es otro tipo de analizador sintáctico de un lenguaje. Un autómata puede también aceptar cadenas de un lenguaje, o árboles o grafos como una gramática.

Definición 4.19 Un *autómata determinista* que acepte cadenas es una tupla de 7 elementos $A = (V, I, \#, Q, q_i, q_a, \delta)$ donde:
V es el conjunto no vacío, finito de símbolos, llamado el vocabulario preestablecido de A.
I es el subconjunto no vacío de V llamado vocabulario de entrada de A.
$\# \in I$ se llama el símbolo blanco.
Q es un conjunto no vacío llamado el estado de A.
$q_i \in Q$ se llama el estado inicial.
$q_a \in Q$ se llama el estado de aceptación.
δ es la correspondencia de $VxQx\{L, R, N\}$ con ella misma, llamada la función de transición de A.

Un autómata es como una ventana que se mueve a lo largo de una cinta de entrada, que inicialmente contiene una cadena infinita de la forma $\#^\infty\alpha\#^\infty$, donde α es una cadena no nula de símbolos de V_I. Inicialmente A se emplaza en algún símbolo no blanco ($no - \#$) de la cadena, y está en el estado q_i sin moverse. Se mueve a lo largo de la cinta de entrada, y reescribe los símbolos, y cambia el estado repitiendo la aplicación de la función de transición δ. Así, si se encuentra en el estado q, sobre el símbolo x, y se ha movido justo en la dirección $d = (I, D, N)$, y $\delta(x, q, d) = (x', q', d')$, entonces reescribe x como x', cambia al estado q' y se mueve en la dirección d'. I significa izquierda, D derecha, y N significa no moverse. Si A llega al estado de aceptación q_a, decimos que A acepta la cadena de entrada X. El conjunto de cadenas de entrada que A acepta es su lenguaje, y se denota como $L(A)$. Se puede ver que los conjuntos de cadenas que son lenguajes de aceptadores son los mismos conjuntos que son lenguajes de gramáticas de tipo sin restricciones, o tipo 0.

Los autómatas pueden ser no deterministas cuando la función de transición δ hace corresponder tripletas (x, q, d) en conjuntos de tripletas (x', q', d'). Se llamará limitada por la cinta (tape-bounded), cuando no pueda reescribir los símbolos # que lea, y cuando llegue a uno tenga que volver en la dirección en que ha venido. Se llamará monotónica cuando no pueda reescribir ni los # ni los que no lo son. Es decir se puede mover a través de los símbolos # pero no podrá reescribirlos. Los lenguajes aceptados por autómatas limitados por cinta, y monotónicos son los mismos, y corresponden a las gramáticas sensibles al contexto.

Un autómata A se llama de estados finitos si no puede reescribir ni símbolos # ni $no - \#$. Esta clase de lenguajes no depende de que el autómata sea limitado por la cinta o no. Se corresponde con las gramáticas de estados finitos, y tampoco dependen de que el autómata sea determinista o no. Algunas de estas características cambian en el caso de vectores y grafos.

Aceptadores de vectores

Un aceptador de vectores se define casi igual que un aceptador de cadenas. Inicialmente se coloca en una posición donde se encuentre un símbolo $no - \#$, y asumimos que la porción α, del vector de entrada, que es $no - \#$ es finita y está conectada. En este caso, al ser el vector de dos dimensiones existen cuatro posibles direcciones de movimiento y no sólo dos, es decir, izquierda, derecha, arriba y abajo, así que usamos $\{I, D, R, B\}$ para definir la función δ.

(a) **Caso de vector rectangular**
El vector de entrada es rectangular. La capacidad del autómata A para aceptar un vector de entrada no depende de la posición inicial, ya que como el vector es rectangular siempre se podrá colocar en una posición de entrada estándar, como por ejemplo la esquina superior izquierda. Debemos notar que esto no ocurrirá igual en el caso de vectores conectados de forma arbitraria. Las clases de lenguajes de vectores rectangulares generados por gramáticas de matrices no son las mismas que las aceptadas por varios tipos de aceptadores de vector. Esto está explicado con mayor detalle en [86].

(b) **Caso general**
La definición de un aceptador en el caso general de vectores es idéntica a la del caso de vectores rectangulares, solo que en este caso no se puede asegurar que el autómata A siempre encontrará la posición inicial estándar. Los lenguajes generados por gramáticas de vector son los mismos que los que aceptan los aceptadores de vector en el caso general. Para ver en mayor detalle este tipo de aceptadores consultar [86].

Aceptadores de grafos

Un aceptador de grafos se coloca en un nodo de su grafo etiquetado de entrada en un estado especial de inicio. Luego se mueve a los nodos vecinos, cambiando el estado y reescribiendo los símbolos según una función de transición. Se dice que acepta el grafo de entrada si llega a un estado especial de aceptación. Aunque la definición de aceptador es similar a la de cadenas y vectores existen algunas diferencias:

- En un grafo no podemos definir el nodo inicial.
- No podemos definir los movimientos del aceptador en términos de arriba, abajo, izquierda o derecha, ya que un node de un grafo puede tener cualquier número de vecinos. De hecho no existe una manera general para el aceptador de distinguir entre los vecinos de un nodo. Se podría dejar al aceptador moverse a una posición vecina de forma aleatoria pero entonces sería imposible volver a la posición previa. Por tanto sería imposible una exploración sistemática del grafo. Para solventar este problem se permite al aceptador que etiquete los nodos adyacentes a la posición actual, para así permitir escoger movimientos que dependan de esas etiquetas.
- Otro problema es cómo añadir nodos $no - \#$ a los grafos de entrada. Para cadenas y vectores definimos una estructura infinita, con un subconjunto finito

de símbolos $no - \#$ y así podemos reescribir símbolos # por símbolos $no - \#$ cuando nos convenga. Pero en el caso de los grafos necesitaríamos definir un grafo completo infinito, con un subgrafo finito de símbolos $no - \#$, en el que cada nodo tuviera infinitos vecinos etiquetados como #. EL problema de esta aproximación es que ahora el aceptador puede ver todo el grafo desde una posición dada. Otra solución alternativa sería permitir al aceptador unir dos nodos en uno o dividir un nodo en dos. Esto permitiría tener siempre un grafo finito.

Las clases de grafos aceptadas por aceptadores de grafos son las mismas que los lenguajes de gramáticas de grafos.

Si nos restringimos a árboles en vez de grafos tenemos que las gramáticas de árbol se pueden ver como equivalentes a una clase de aceptadores llamada autómata de árbol. Existe otra clase restringida de aceptadores de grafos que necesita que los grafos tengan un grado de límite, por ejemplo r, y que los arcos a cada nodo tengan etiquetas de un conjunto finito $e_1, \ldots, e_n$, que representen todas las direcciones posibles de movimiento. Esta aproximación generaliza el caso de cadenas y vectores, donde cada nodo puede tener dos o cuatro direcciones de movimiento respectivamente.

Dada una gramática que describa los patrones de interés que tengamos y una representación simbólica x de un patrón desconocido, el problema de reconocer el patrón sintácticamente pasa por hacer el análisis sintáctico de x en función de G. Analizar sintácticamente significa decidir si $x \in L(G)$ o no, y construir el árbol sintáctico de x de acuerdo con G si $x \in L(G)$. En la Fig. 4.1 se presenta un esquema gráfico del proceso.

La teoría formal de grafos nos dice que el análisis sintáctico de gramáticas libres de contexto es un problema no determinista, y que los únicos algoritmos de análisis sintáctico que funcionan en tiempo y espacio lineal son los existentes para la subclase de los lenguajes regulares dentro de la clase de lenguajes libres de contexto. Para gramáticas sin restricciones el análisis sintáctico no es completamente decidible. Es decir no existe un algoritmo general que funcione para un gramática sin restricciones que garantice su terminación en tiempo y espacio finitos para cualquier entrada. Sólo si $x \in L(G)$ podemos garantizar que el algoritmo terminará con el resultado correcto, pero el algoritmo puede entrar en un bucle infinito si $x \notin L(G)$.

En general los algoritmos de análisis sintáctico pueden clasificarse en:

- **De análisis ascendente** o (bottom-up): Tratan de construir el árbol sintáctico de abajo a arriba, es decir de las hojas a la raíz.
- **De análisis descendente** o (top-down): Tratan de construir el árbol sintáctico de arriba a abajo, es decir, de la raíz a las hojas.
- También existen otros métodos que combinan las dos aproximaciones.

4.4.1 Algoritmo de análisis sintáctico: Earley

Este algoritmo fue presentado por primera vez en [37]. Una descripción de su pseudocódigo puede verse en el algoritmo presentado en el Alg. 4.4.1. Este algoritmo recibe como parámetros una cadena $x_1, \ldots, x_n$ de símbolos terminales y una gramática libre

de contexto $G = (N, T, P, S)$, y genera unas listas $L(0), L(1), \ldots, L(n)$. Es decir una lista por cada símbolo terminal de entrada y una lista $L(0)$. Inicialmente estas listas están vacías. Un elemento de la lista es de la forma $(A \to \alpha \bullet \beta, j)$ donde:

(a) $A \to \alpha\beta$ es una producción; $A \in N$; $\alpha, \beta \in V^*$

(b) α es la primera parte de la parte derecha de la regla. Se ha reconocido en la cadena de entrada en el momento que el elemento de la lista se genera.

(c) j es un puntero a la lista $L(j)$ o, de forma equivalente, a la posición de entrada j. La parte de entrada correspondiente a α comienza en la posición $j+1$.

(d) $\bullet$ indica el punto de la cadena que estamos analizando en el momento actual.

Se deduce del algoritmo que siempre que $(A \to \alpha \bullet \beta, j)$ está en $L(i)$ entonces $j \leq i$. Además si $(A \to \alpha \bullet \beta, j)$ está en $L(i)$, entonces $\alpha \overset{*}{\to} x_{j+1}x_{j+2}\ldots x_i$. Si substituimos S por A, ϵ por β, 0 por j y n por i observamos que si $(S \to \alpha\bullet, 0)$ está en $L(n)$ entonces $S \overset{*}{\to} x_1x_2\ldots x_n$ o, lo que es lo mismo, $x \in L(G)$. Esto significa que para decidir si $x \in L(G)$ o no generaremos de forma secuencia las listas $L(0), L(1), \ldots, L(n)$. Si el elemento de la lista $(S \to \alpha\bullet, 0)$ está en $L(n)$ podemos concluir que $x \in L(G)$, y de lo contrario $x \notin L(G)$. La complejidad en tiempo y espacio del algoritmo de parser Earley es de $O(n^3)$ y $O(n^2)$ respectivamente. Existen algoritmos para construir uno o todos los árboles de derivación posibles a partir de $L(0), L(1), \ldots, L(n)$ si $x \in L(G)$.

Algoritmo 4.4.1 Parser: Earley

---

entrada: $x = x_1 \ldots x_n \in T^*$ y una gramática libre de contexto $G = (N, T, P, S)$
salida: $L(0), L(1), \ldots, L(n)$
método:
// Inicialización: Construcción de $L(0)$
Para cada producción $S \to \alpha$ en P añadir $(S \to \bullet\alpha, 0)$ a $L(0)$; $\alpha \in V_+$
Repetir
  Para cada item $(A \to \bullet B\beta, 0)$ en $L(0)$ y para cada producción $B \to \varphi$ en P añadir $(B \to \bullet\varphi, 0)$ a $L(0)$; $A, B \in N$; $\beta\ in V^*$; $\varphi \in V^+$
Hasta que: No se puede añadir ningún item nuevo a $L(0)$
// Bucle principal: Creación de $L(1), \ldots, L(N)$
Para $i = 1$ hasta n **hacer**
  // Análisis lexicográfico
  Para cada item $(A \to \alpha \bullet a\beta, j)$ en $L(i-1)$ donde $a = x_i$ añadir $(A \to \alpha a \bullet \beta, j)$ a $L(i)$; $A \in N$; $a \in T$; $\alpha, \beta \in V^*$;
  Repetir
    // subrutina completar
    Para cada item $(B \to \varphi\bullet, j)$ en $L(i)$ y cada it em $(A \to \alpha \bullet B\beta, k)$ en $L(j)$ añadir $A \to \alpha B \bullet \beta, k$ a $L(i)$;$A, B \in N$; $A, B \in N$; $\alpha, \beta \in V^*$; $\varphi \in V^+$;
    // subrutina predecir
    Para cada item $(A \to \alpha \bullet B\beta, j)$ en $L(i)$ y cada producción $B \to \varphi$ en P añadir $B \to \bullet\varphi, i$ a $L(i)$; $A, B \in N$; $\alpha, \beta \in V^*$; $\varphi \in V^+$;
  Hasta que: no se puede añadir ningún nuevo item a $L(i)$
FinPara

4.4.2 Algoritmo de análisis sintáctico: Earley con corrección de errores

El algoritmo presentado en el apartado anterior no puede ser utilizado con patrones que tengan distorsiones. Simplemente con que un símbolo de la cadena sea diferente el parser la rechazará. Este problema puede solucionarse con la distancia de cadenas presentada en el Capítulo 2 $d(x, y)$. Esta distancia que era presentada entre dos cadenas puede generalizarse como una distancia entre una cadena y el lenguaje generado por una gramática $d(x, L(G))$. De este modo la extensión del algoritmo de Earley para hacer el análisis sintáctico con corrección de errores quedaría de la siguiente forma.

Sean $G_1, \ldots, G_N$ un conjunto de gramáticas que representan un patrón cada una una. Dado un patrón, x, desconocido y distorsionado, se calculan las N distancias $d(x, L(G_1), \ldots, d(x, L(G_N)$ y se escoge la mínima, $d(x, L(G_i)$. Un valor de corte ψ puede definirse para decidir si x pertenece a la clase i si y sólo si $d(x, L(G_i) \leq \psi$.

En un parser con corrección de errores se utilizan las mismas operaciones que para el cálculo de la distancia de cadenas presentado en el Capítulo 2. Es decir, la inserción substitución o eliminación de símbolos terminales de la gramática. Para simplificar consideraremos que el coste de cualquiera de estas operaciones será de uno. Dada una gramática G libre de contexto definiremos:

$$d(x, L(G)) = min\{(d(x, y)|y \in L(G)\}) \tag{4.3}$$

Es evidente que si $x \in L(G)$ entonces $d(x, L(G)) = 0$. Pero aunque esta definición es muy simple existe el problema de cómo calcular $d(x, L(G))$ dado que $L(G)$ puede ser infinita. Para solucionar esto debemos primero definir el concepto de *cobertura de una gramática*. Sea $G = (N, T, P, S)$ una gramática libre de contexto, definiremos su cobertura $G' = (N', T', P', S')$ siguiendo los siguientes pasos:

1. $N' = N \bigcup \{S\} \bigcup \{E_a | a \in T\}$. S' es el nuevo símbolo inicial de la gramática cobertura. Cada E_a es un no-terminal nuevo que representa el terminal a de la gramática original G.

2. P' se obtiene con los siguiente pasos:
   (2.1) Para cada producción $p \in P$ de la forma $A \rightarrow \alpha_0 a_1 \alpha_1 a_2, \ldots, a_m \alpha_m$; $m \geq 0$; $\alpha_j \in N^*$; $a_i \in T$ se añadir la producción $A \rightarrow \alpha_0 E_{a_1} \alpha_1 E_{a_2}, \ldots, E_{a_m} \alpha_m$ a P'. Esto significa simplemente la sustitución, en cada producción, de los terminales a por sus correspondientes no-terminales E.
   (2.2) Para cada símbolo $a \in T$ se añaden las siguientes producciones a a P'.
      a) $E_a \rightarrow a$
      b) $E_a \rightarrow b$ para todo $b \in T$, $b \neq a$.
      c) $E_a \rightarrow \epsilon$.
      d) $E_a \rightarrow bE_a$ para todo $b \in T$, también $b = a$.
      e) $S' \rightarrow S'a$.
      f) $S' \rightarrow S$.

Así las producciones generadas por a) corresponden a las que no tienen errores, las generadas por b) y c) son la sustitución y borrado respectivamente. Las inserciones

al comienzo y mitad de una palabra están modelizadas por d), mientras que e) y f) modelizan inserciones al final. Por tanto son las producciones de la b) a la f) las que llamamos producciones de error. Una cobertura G' de una gramática G es otra gramática que genera siempre T^* independientemente de G, es decir $L(G') = T^*$. De este modo la gramática G' es lo suficientemente potente para poder simular cualquier secuencia de transformación de error que pueda afectar a una palabra de $L(G)$.

El algoritmo de Earley con corrección de errores quedaría como se ve en Alg. 4.4.3. Este algoritmo se presentó por primera vez en [5]. Puede apreciarse que la entrada y salida del algoritmo son las mismas que para la versión sin corrección de errores. Los items de las listas $L(i)$, para $i = 0, \ldots, n$ son de la forma: $(A \to \alpha \bullet \beta, j, t)$ donde A, α, β y j tienen el mismo significado que en el algoritmo de Earley sin corrección de error, y t indica el número mínimo de transformaciones de error que se necesita para derivar α en $x_{j+1}, x_{j+2}, \ldots, x_i$. La operación "añadir $(A \to \alpha \bullet \beta, j, t)$ a $L(i)$" se define de la siguiente forma:

Algoritmo 4.4.2 añadir $(A \to \alpha \bullet \beta, j, t)$ a $L(i)$

Si no hay un item $(A \to \alpha \bullet \beta, j,)$ en $L(i)$ **Entonces**
  añadir$(A \to \alpha \bullet \beta, j, t)$ a $L(i)$
Sino {//Cuando si que hay un item $A \to \alpha \bullet \beta, j, t')$ en $L(i)$}
  Si $t' > t$ **Entonces**
    reemplaza $(A \to \alpha \bullet \beta, j, t')$ por $(A \to \alpha \bullet \beta, j, t)$
  Fin Si
Fin Si

Esto quiere decir que si se encuentran dos items $(A \to \alpha \bullet \beta, j, t)$ y $(A \to \alpha \bullet \beta, j, t')$ que difieren sólo en el número de transformaciones de error, existen dos derivaciones diferentes en el análisis sintáctico. En este caso se escogerá la alternativa con un número menor de transformaciones de error.

Se puede ver que si $(A \to \alpha \bullet \beta, j, t)$ está en $L(i)$, entonces $\alpha \overset{*}{\to} a_{j+1}, a_{j+2}, \ldots, a_i$ con t producciones de error, y no existe otra derivación con menos producciones de error. De ello se deduce que $d(x, L(G)) = t$ si hay un item $(S \to \alpha \bullet, 0, t$ en $L(n)$. Eso quiere decir que en la aplicación del algoritmo de análisis sintáctico generaremos $L(0), L(1), \ldots, L(n)$ y buscaremos un item $S \to \alpha \bullet, 0, t)$ en $L(n)$. La existencia de tal item está garantizada y sabemos que $d(x, L(G)) = t$. La complejidad en tiempo y espacio del algoritmo es $O(n^3)$ y $O(n^2)$ respectivamente. Igual que en la versión original del algoritmo el árbol de derivación de una cadena x respecto a una cobertura de gramática G' se puede construir a partir de las listas $L(0), L(1), \ldots, L(n)$. Este árbol de derivación contendrá todas las producciones de error necesarias para generar x. Si las producciones de error se borran obtendremos la versión correcta de x, es decir $y \in L(G)$ que satisface la ecuación 4.3. Así vemos que los parser con corrección de error no sólo sirven para clasificar patrones distorsionados sino que también son útiles para corregir errores de forma explícita y para dar una interpretación estructural de las cadenas de entrada distorsionadas.

Algoritmo 4.4.3 Parser: Earley con corrección de errores

entrada: $x = x_1 \dots x_n \in T^*$ y una cobertura de u na gramática libre de contexto $G' = (N', T', P', S')$
salida: $L(0), L(1), \dots, L(n)$
método:
// Inicialización: Construcción de $L(0)$
Para cada producción $S' \to \alpha$ en P' añadir $(S' \to \bullet\alpha, 0, 0)$ a $L(0)$; $\alpha \in V'^*$
Repetir
// Subrutina predecir de $L(0)$
Para cada item $(A \to \bullet B\beta, 0, t)$ en $L(0)$ y para cada producción $B \to \varphi$ en P' añadir $(B \to \bullet\varphi, 0, 0)$ a $L(0)$; $A, B \in N'$; $\beta, \varphi \in V'^*$;
// Subrutina completar de $L(0)$
Para cada item $(B \to \varphi\bullet, 0, t)$ y $(A \to \alpha \bullet B\varphi, 0, t')$ en $L(0)$ añadir $(A \to \alpha B \bullet \varphi, 0, t'')$ a $L(0)$, donde

$$t'' = \begin{cases} t + t' + 1 & si\, B \to \varphi\, es\, una\, producción\, de\, error \\ t + t' & sino. \end{cases}$$

$A, B \in N'$;$\alpha, \beta, \varphi \in V'^*$
Hasta que: No se puede añadir ningún item nuevo a $L(0)$
// Bucle principal: Creación de $L(1), \dots, L(n)$
Para $i = 1$ hasta n **hacer**
// subrutina análisis lexicográfico
Para cada item $(A \to \alpha \bullet a\beta, j, t)$ en $L(i-1)$ donde $a = x_i$ añadir $(A \to \alpha a \bullet \beta, j, t)$ a $L(i)$; $A \in N'$; $a \in T'$; $\alpha, \beta \in V'^*$;
Repetir
// subrutina completar
Para cada item $(B \to \varphi\bullet, j, t)$ en $L(i)$ y cada item $(A \to \alpha \bullet B\beta, k, t')$ en $L(j)$ añadir $(A \to \alpha B \bullet \beta, k, t'')$ a $L(i)$, donde

$$t'' = \begin{cases} t + t' + 1 & si\, B \to \varphi\, es\, una\, producción\, de\, error \\ t + t' & sino. \end{cases}$$

$A, B \in N'$; $\alpha, \beta, \varphi \in V'^*$;
// subrutina predecir
Para cada item $(A \to \alpha \bullet B\beta, j, t)$ en $L(i)$ y cada producción $B \to \varphi$ en P' añadir $(B \to \bullet\varphi, j, 0)$ a $L(i)$;
$A, B \in N'$; $\alpha, \beta, \varphi \in V'^*$
Hasta que: no se puede añadir ningún nuevo item a $L(i)$
FinPara

4.5 Inferencia gramatical

Una gramática puede usarse para describir la sintaxis de un lenguaje o la estructura de un patrón, pero también para caracterizar una fuente sintáctica que genere todas las sentencias, finitas o infinitas, de un lenguaje, o los patrones de una determinada clase. Esto hace que sea deseable poder obtener esta gramática de una manera automática a partir de la muestra de frases del lenguaje o del conjunto de muestras de la clase de patrones. El problema de aprender una gramática de forma automática a partir de un conjunto de muestra se llama *inferencia gramatical*. La inferencia gramatical se ha aplicado en diversas áreas como: el reconocimiento de patrones, la recuperación de información, el diseño de lenguajes de programación, la traducción y compilación, los lenguajes gráficos, la comunicación persona-máquina, y la inteligencia artificial.

El problema de la inferencia gramatical, explicado a un nivel más formal, se refiere básicamente a los procesos que se necesitan para inferir las reglas sintácticas de una gramática G desconocida, a partir de un conjunto finito de frases o cadenas del lenguaje $L(G)$ generado por G. A veces, también, a partir del lenguaje complementario de $L(G)$, es decir de un conjunto de contraejemplos. Así, el proceso de inferencia, obtiene las reglas sintácticas que describen el conjunto finito de cadenas de $L(G)$ que se le ha dado como muestra, y que permiten además predecir otras cadenas que tengan una estructura de la misma naturaleza. Normalmente se define una medida de bondad de la gramática inferida, en función de la complejidad de las reglas de la misma.

La inferencia gramatical es parte del campo del aprendizaje automático, concretamente pertenece a la inferencia inductiva, es decir sistemas que tratan de extraer reglas generales a partir de ejemplos. En el caso concreto de la inferencia gramatical estas reglas son reglas de reescritura en el sentido de la Teoría de Lenguajes Formales, o una extensión natural de ellas.

En el campo del reconocimiento de patrones, la inferencia gramatical da métodos automáticos para aprender los modelos de las clases de patrones. Es una parte necesaria y útil del reconocimiento sintáctico y estructural de patrones, y ha producido herramientas y metodologías útiles para el aprendizaje estructurado.

Antes de seguir con la explicación de la inferencia gramatical se necesitan definir algunos conceptos:

Definición 4.20 Una *secuencia de información* de un lenguaje L, se denota como $I(L)$, y es una secuencia de cadenas del conjunto $\{x|x \in L\} \cup \{x|x \in T^* - L\}$. La *secuencia de información positiva* $I^+(L)$ es la secuencia de cadenas del conjunto $\{x|x \in L\}$, es decir sólo contiene cadenas que pertenecen a L. La *secuencia de información negativa* $I^-(L)$ es una secuencia de cadenas del conjunto $\{x|x \in T^* - L\}$, es decir sólo contiene cadenas formadas por terminales que forman el lenguaje L pero que no pertenecen al mismo.

Definición 4.21 Una *muestra* de un lenguaje L, denotada como $M_t(L)$, se define como $M_t(L) = M^+(L) \cup M^-(L)$. Donde $M_t^+ = \{+x_1, \ldots, +x_t\}$ es la muestra positiva, y $M_t^- = \{-x_1, \ldots, -x_t\}$ es la muestra negativa.

Dado un conjunto finito de t secuencias, M_t, llamado muestra, la inferencia gramatical se hace la siguiente pregunta: £'Es posible encontrar una gramática regular

G tal que $M_t \subset L(G)$? Como se sabe el problema de la inferencia gramatical tiene infinitas soluciones. Una es la trivial, la gramática universal G_u, que acepta el lenguaje X^*, siendo X el alfabeto en el que está definida M_t. Le podemos añadir a X tantos símbolos nuevos como queramos y seguiremos teniendo la gramática universal. Otra solución es la llamada, *gramática canónica maximal*, $GCM(M_t)$, que es una de las gramáticas para las que el lenguaje generado es idéntico a M_t. Aquí también podemos añadir cuantas transiciones queramos a la gramática, que tengan símbolos que deban o no perteneces a X, sin salirnos del conjunto de soluciones. Esto nos permite tener un conjunto infinito de posibilidades, y para evitar esto se impone la condición de que el conjunto de muestra sea *completo*.

Definición 4.22 Una secuencia de información es *completa* si:
1. $I^+(L)$ contiene todas las cadenas de L.
2. $I^-(L)$ contiene todas las cadenas en T^* que no son de L.

Definición 4.23 Una muestra M_t se dice que es *estructuralmente completa* (o simplemente completa) con respecto a una gramática $G = (N, T, P, S)$ si cada producción definida en G se usa en la generación de, al menos, una cadena de M^+:
(a) $M_t \subset L(G)$
(b) N es el alfabeto en que M_t está escrita.
(c) Cada regla P se usa por lo menos una vez en la generación de cadenas en M_t.

Dada esta condición el problema de la inferencia gramatical es:

Definición 4.24 Para una muestra M_t, la *inferencia gramatical* consiste en encontrar una gramática, o todas las posibles, tales que:
(a) $M_t \subset L(G)$
(b) M_t es completa con respecto a G.

Con esta restricción se pueden construir algoritmos para soluciones enumeradas, particularmente para el caso regular, pero el número de soluciones será siempre muy elevado en relación con la medida de la muestra. Es decir, dada una muestra obtendremos un conjunto amplio de gramáticas que la generan. El problema es que en un caso práctico para reconocimiento de patrones no podemos quedarnos con este conjunto de posibles soluciones. Lo que necesitamos es seleccionar de entre todas las posibilidades una sola. Para ello necesitamos métodos heurísticos, que nos permitan primero reducir el número de soluciones a ser consideradas, y en un segundo lugar, seleccionar, de entre el subconjunto preseleccionado una sola. Decimos que son heurísticas porque no podemos asegurar con certeza que este proceso seleccione la gramática que optimice el criterio.

Dado que un método práctico de inferencia gramatical es un algoritmo para seleccionar una gramática de entre un conjunto numeroso de soluciones, y que la evaluación de sus cualidades es bastante subjetiva, existen varios métodos para tal proceso difíciles de comparar entre sí de entre los que el usuario debe escoger el que mejor se adapte a su problema.

Definición 4.25 Sea $z \in T^*$, la *k-cola* de z respecto a $M \in 2^{T^*}$ se denota como $g(z, M, k)$ y se define como $g(z, M, k) = \{x \in T^* | zx \in M \ y \ |x| \leq k\}$, donde $|x|$ es la longitud de la cadena x, $k \geq 0$.

Definición 4.26 Sea f una función que pone en correspondencia el conjunto de no terminales de la gramática G_2 con el de la gramática G_1. Además pone en correspondencia el no terminal inicial de G_2 con el de G_1, y f se aplica a poner en correspondencia las producciones de G_2 con las de G_1. Entonces se dice que G_1 *cubre* a G_2 y $L(G_2) \subseteq L(G_1)$.

Definición 4.27 Una gramática G_i es *compatible* con una muestra M_i si $L(G_i)$ contiene todas las cadenas de M_i^+, y no contiene ninguna de las cadenas del ejemplo negativo M_i^-.

Definición 4.28 Una clase de gramáticas C es *admisible* si:
1. C es numerable.
2. Para $x \in T^*$, es decidible si x pertenece o no a $L(G)$ para cualquier gramática $G \in C$.

Definición 4.29 Todas las gramáticas sensibles al contexto, libres de contexto y regulares son numerables y recursivas y por tanto decidibles.

Definición 4.30 Una clase de lenguajes $L(G)$ se dice que es *identificable en el límite* si hay un algoritmo de inferencia Z tal que para cualquier $G \in C$ y cualquier secuencia de información completa $I(L(G))$ existe un τ tal que:
(1) $G_t = G_\tau$, para $t > \tau$, donde $G_t = Z(M_t, C)$ y $G = Z(M_\tau, C)$
(2) $L(G_t) = L(G)$.

Definición 4.31 Un algoritmo Z se dice que *aproxima* a una gramática G si se cumplen las dos condiciones siguientes:
1. Para cualquier $x \in L(G)$ hay un τ tal que $t > \tau$ implica que $x \in L(G_t)$, donde $G_t = Z(M_t, C)$.
2. Para cualquier G' tal que $L(G') - L(G) \neq \emptyset$, hay un τ tal que $t > \tau$ implica $G_t \neq G'$.

Se dice que Z *aproxima fuertemente* a G si, además de las condiciones 1 y 2 se cumple que existe un H tal que $L(H) = L(G)$ y que para un número finito t, $G_t = H$. Intuitivamente, un algoritmo Z se puede usar para identificar a G si finalmente supone una sola gramática y esa gramática genera exactamente $L(G)$. Pero eso no implica que Z escoja al final una gramática y pare de considerar nuevos datos. Si Z puede solamente garantizar que al final rechazará cualquier gramática que no produzca $L(G)$ entonces decimos que $L(G)$ es aproximable.

Definición 4.32 Una *secuencia de información estocástica positiva* de un lenguaje estocástico $L_s = L(G_s)$, se denota por $I(L_s)$, y se define como una secuencia de cadenas, cada una de las cuales es una variable aleatoria generada estadísticamente por la gramática estocástica G_s. Una *muestra positiva estocástica* de un lenguaje estocástico $L_s = L(G_s)$, se denota por $M_t^+(L_s)$, y se define como el conjunto $\{m_1, \ldots, m_t\}$, donde $m_1, \ldots, m_t$ son cadenas (variables aleatorias) generadas por la gramática estocástica G_s.

Gold presentó un estudio de identificación de lenguaje en el límite por enumeración. Su modelo de identificación de lenguaje consistía en tres componentes básicos:

1. Una clase de gramáticas numerable, C. Suponiendo que se escoja una gramática $G \in C$. Una máquina de inferencia gramatical determina, basándose en una secuencia de información del lenguaje L generado por G, qué gramática es de las de C.

2. Un método para presentar la información. Asume que el tiempo se cuantifica y empieza en un tiempo finito, es decir $t = 1, 2, 3, \ldots$. En cada tiempo t, la máquina de inferencia gramatical se presenta con una cadena de entre las de la secuencia de información de L. La presentación del texto y la presentación del informante son los dos métodos básicos de presentación de la información considerados. Una presentación del informante de L es una secuencia de información completa de L, mientras que una presentación de texto de L es una secuencia de información positiva y completa de L.

3. Un algoritmo de inferencia gramatical. Un algoritmo de inferencia gramatical Z puede hacer una suposición en la base de M_t tal que $Z(M_t, C) = G_t$ para $G_t \in C$. Todos los algoritmos se construyen basándose en la numerabilidad de varias clases de gramáticas.

Las aplicaciones reales de inferencia gramatical son limitadas por las siguientes razones:

- Dado un conjunto finito de patrones ejemplo existen infinitas gramáticas que pueden generar un lenguaje que contenga ese conjunto de patrones como subconjunto. Por ello se necesitan restricciones adicionales para escoger entre las gramáticas que dan una solución potencial. Estos criterios son difíciles de encontrar.

- Las gramáticas puramente formales como las regulares o las libres de contexto no suelen ser lo suficientemente potentes para trabajar con aplicaciones reales. Por ello se necesita añadir heurísticas, sin embargo éstas suelen estar más allá del ámbito de la mayoría de los algoritmos de inferencia.

- El diseño de un sistema de reconocimiento sintáctico de patrones necesita la construcción de una gramática, pero también seleccionar las primitivas del patrón, seleccionar el modelo de gramática, si es de cadena, de árbol o de grafo, con producciones regulares o libres de contexto. Aunque el proceso de inferir la gramática pueda llegar a hacerse de forma automática el resto deberá realizarse de forma manual usando la experiencia y la intuición.

Aunque en el paradigma puro del reconocimiento sintáctico de patrones la situación de la inferencia gramatical está clara, en la práctica las cosas son diferentes:

- Una hipótesis sintáctica pura es una situación ideal, pero se sabe que los patrones reales no son libres de contexto ni regulares, y no sería realista esperar tener un buen rendimiento en situaciones reales a partir de modelos clásicos estrictos.

- Aunque supusiéramos el modelo clásico estricto como válido la inferencia gramatical es muy difícil. La formulación teórica del problema no es de gran ayuda en la práctica, y el programador debe tratar un problema exponencial.

Por todo ello esta técnica tiene problemas para ser utilizada realmente en el reconocimiento de patrones, y muchas veces en este campo se opta por definir las gramáticas a mano para obtener una mayor precisión. Por otro lado muchas veces la utilidad de una gramática está en poder deducirla automáticamente para, por ejemplo, poder modelizar la clase de un conjunto de patrones, por ello, en ocasiones en que un conocimiento del entorno de aplicación permite realizar algunas simplificaciones en el proceso, esta inferencia es útil.

La inferencia gramatical puede clasificarse según varios puntos de vista. Por ejemplo puede decirse que existen dos tipos de inferencia gramatical. Uno orientado a inferir una gramática desde el punto de vista teórico, donde lo interesante es el proceso de aprendizaje abstracto y la convergencia en una muestra infinita de sentencias. El otro está orientado a resolver un problema práctico concreto. Se tienen unos datos en un formato concreto y se sabe que tipo de reglas se quieren generar,incluso, a veces, se restringe más el problema dándole información extra.

Otra forma de clasificación de la inferencia gramatical puede hacerse teniendo en cuenta qué tipo de gramática se va a generar, es decir, libre de contexto, regular, etc.

Una tercera clasificación se refiere al tipo de algoritmo diseñado. Si es una búsqueda en un espacio estado cuya estructura se usa para abolir la fuerza bruta en la enumeración, si es una optimización de parámetros con un método de gradiente, si es una solución de acceso directo con una heurística *ad hoc*. Esta última clasificación nos muestra que, el principal problema que se encontrará el diseñador de un algoritmo de inferencia gramatical, es evitar el problema de la complejidad exponencial inherente a la aplicación, con respecto a la medida de la muestra, y que para solucionarlo necesitará utilizar heurísticas implícitas o explícitas.

En el área de las gramáticas regulares, la inferencia gramatical se puede resolver claramente aunque no da algoritmos constructivos. Serán heurísticas basadas en las propiedades algebraicas.

Vamos a presentar algunos métodos prácticos de inferencia gramatical regular, para una explicación más detallada de los mismos ver [74].

Tenemos el *método del sucesor* que consiste en crear un autómata de la siguiente manera. Se crea un estado para cada símbolo del alfabeto que ocurre en la muestra, y un estado inicial. Para el estado inicial miramos por que símbolos puede empezar una cadena y generamos transiciones, al recibir los mismos, a sus estados correspondientes. Para cada símbolo miramos que símbolos pueden ir detrás de él y generamos transiciones del estado que representa a ese símbolo a los estados que representan a sus sucesores. Para finalizar miramos por qué símbolos puede terminar una cadena y hacemos que sus estados correspondientes sean estados finales.

El *Método de Pao-Carr* consiste en definir un autómata finito utilizando un "maestro" que sabe *a priori* la gramática que se ha de inferir y al que se le pueden hacer preguntas sobre la misma en el proceso de inferencia.

El *Algoritmo de* uv^kw consiste en buscar repeticiones de uno o más símbolos en el conjunto de cadenas de la muestra para generarlos en forma de sucesivas pasadas por un bucle en el autómata representativo de la gramática. Primero se escriben las cadenas de la muestra en forma de expresión regular. Luego se buscan todas las posibles repeticiones de uno o varios símbolos en cada uno de los elementos de la muestra, y se selecciona la que se repite más generando un nuevo símbolo que la representa.

A continuación se reescribe esa repetición por un nuevo símbolo, y se vuelven a buscar repeticiones y se vuelve a aplicar el proceso hasta que ya no encontramos más repeticiones. Este método se adapta muy bien a estructuras repetitivas.

El *método de las k colas* consiste en encontrar la relación de k-equivalencia entre los estados de la GCM o cualquier gramática canónica. Este método no se adapta muy bien a los problemas del reconocimiento de patrones aunque se ha usado en la práctica probablemente por ser fácil de hacer funcionar.

El *método del clustering de cola* es una generalización del método anterior.

El *método de la inferencia de lenguajes reversibles.* Estos lenguajes son los que el autómata que los define y su inverso son deterministas. El inverso a un autómata es otro autómata que tiene como estado final el estado inicial del primero y como estado inicial el final, y las transiciones las tiene a la inversa.

4.6 Modelización y reconocimiento de símbolos texturados

Una vez introducidos los principios teóricos en los que se sustenta este trabajo vamos a explicar cómo realizamos la modelización, y posteriormente el reconocimiento de los símbolos que hemos llamado *símbolos texturados*, y que son el objeto del trabajo de esta tesis. Primero explicaremos cómo se pueden representar estos símbolos en el Apartado 4.6.1, y de ahí deduciremos como modelizarlos ver Apartado 4.6.2. Una ver presentado el modelo explicaremos como se puede inferir el mismo a partir de un ejemplo dado, ver Apartado 4.6.3. A continuación, y debido a las distorsiones que nos pueden aparecer en los casos prácticos que queremos analizar presentaremos una modelización que tiene en cuenta estas deformaciones, o lo que es lo mismo, presentaremos una modelización con corrección de errores, ver Apartado 4.6.4. Por último veremos como, dada una representación de un símbolo, podemos analizar un posible símbolo de entrada y ver si pertenece a la misma clase, ver Apartado 4.6.5.

4.6.1 Definición del problema

Dentro del reconocimiento de símbolos podemos encontrar básicamente dos tipos de ellos. El primero es el que representa el símbolo por medio de un patrón prototipo, es decir un conjunto de primitivas emplazadas según una estructura fija. El segundo es el basado en texturas estructuradas, es decir un conjunto de primitivas que se emplazan siguiendo unas reglas de emplazamiento, pero el número de instancias de una primitiva puede variar de un ejemplo a otro del mismo símbolo. En la Fig. 4.6 vemos cuatro tipos de texturas estructuradas que nos podemos encontrar. La primera en la Fig. 4.6(a) consta de un solo tipo de primitiva, un rectángulo que se emplaza siguiendo una regla de emplazamiento que refleja vecindades en dos dimensiones. La segunda en la Fig. 4.6(b) consta de dos primitivas que se emplazan según unas reglas que también reflejan vecindades en dos dimensiones. La tercera en la Fig. 4.6(c) consta de una sola primitiva que se emplaza siguiendo una regla de emplazamiento que determina una vecindad en una sola dirección que es relativa a la vecindad de sus vecinos. La última en la Fig. 4.6(d) muestra una textura formada por una sola

primitiva que se emplaza siguiendo una regla que representa una vecindad en una sola dirección pero que esta vez es absoluta y no depende de la de los vecinos.

Como podemos observar, los símbolos texturados necesitan ser representados por un modelo que permita reflejar la estructura de los mismos. Es decir los elementos que los componen y las vecindades entre ellos. Dado que estas vecindades pueden ser múltiples necesitamos una estructura que pueda definir n-dimensiones, es decir un grafo. Además como los elementos que forman el símbolo texturado pueden ser de varios tipos necesitamos poder determinar para cada vecino qué tipo de elemento es, es decir necesitamos etiquetar los nodos que los representan con su tipo correspondiente, además las aristas que reflejan la vecindad entre elementos también nos están indicando qué tipo de vecindad tienen los mismos, y debemos poder distinguirlas unas de otras, por tanto las aristas del grafo tendrán que estar también etiquetadas. Por todo ello la estructura natural para representar una ocurrencia de un símbolo texturado finito es un grafo como el definido en el apartado 1.5.3 $X = (K, R)$, que en el caso concreto que nos ocupa es un grafo de adyacencia etiquetado y con atributos $X = (K, R, FR, LR)$, donde:

- K es el conjunto de nodos del grafo que representan las primitivas que forman el símbolo texturado.
- R es el conjunto de aristas del grafo, que representan las relaciones de vecindad de estos nodos.
- FK es una función que le asigna a cada nodo del grafo $k \in K$ un atributo. Este atributo es una cadena cíclica que representa la secuencia de segmentos que forman la primitiva, en este caso un polígono. Así FK se define como $FK : K \to E^*$, donde E es el conjunto de segmentos del documento vectorizado.
- FR es una función que asigna a cada arista del grafo (k, k') como atributo el segmento de recta que conecta los centros de los polígonos vecinos representados por k y k'. Así FR se define como $FR : R \to C'$, donde C' es el conjunto de segmentos conectando los polígonos vecinos del grafo de adyacencia de regiones $\mathcal{H}$.
- LK es la función que asigna etiquetas a los nodos del grafo.
- LR es la función que asigna etiquetas a las aristas del grafo.

Por medio de este grafo representaríamos sólo una ocurrencia del símbolo texturado, pero lo que queremos realmente es representar la clase de este símbolo texturado. Es decir representar la estructura que ha de tener el símbolo sin restringir su tamaño, contorno o número de repeticiones de sus primitivas. Por ello vemos que no podemos representarlo simplemente con un grafo que sería sólo una de las posibilidades de ese símbolo concreto, sino que necesitamos una estructura que permita definir esa flexibilidad. La estructura que necesitamos ha de poder representar una clase de grafos de adyacencia de regiones con atributos, y como hemos visto en este capítulo tal estructura es una gramática de grafo con atributos.

Nuestro modelo para la representación de símbolos texturados es una *gramática de grafo de atributos con etiquetas en los nodos y las aristas* que se define como una

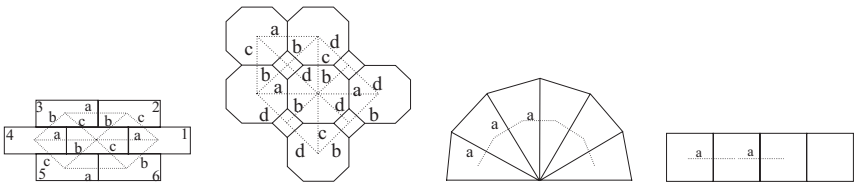


Figura 4.6: Símbolos texturados de diferentes tipos: (a) Con 1 primitiva y emplazamiento en 2D. (b) Con 2 primitivas y emplazamiento en 2D. (c) Con 1 primitiva y emplazamiento en 1D. (d) Con 1 primitiva y emplazamiento en 1D.

variante de la Def. 4.16, donde añadimos una función Ω para asignar etiquetas a las aristas, y otra F en las producciones para determinar cómo se transfieren los atributos de la parte derecha de una regla a la parte izquierda. La definición de gramática de grafo con atributos quedaría en nuestro caso como una 5-tupla $G = (\Sigma, \Delta, \Omega, P, S)$ donde:

- Σ es el alfabeto de las etiquetas de los nodos no-terminales.
- Δ es el alfabeto de las etiquetas de los nodos terminales.
- Ω es el alfabeto de las etiquetas de las aristas terminales.
- P es el conjunto finito de las producciones de la gramática o las reglas de reescritura.
- S es el conjunto de grafos iniciales, que normalmente consiste en un nodo con una etiqueta no-terminal.

Una producción P de la gramática G es una 4-tupla $P = (l, r, T, F)$, donde:

- l es el grafo de la parte izquierda.
- r es el grafo de la parte derecha.
- T es la transformación de inmersión $T = \{(n, n') | n \in V_l, n' \in V_r\}$.
- F el conjunto de funciones de transferencia de atributos, para las aristas y los nodos en h_r.

La derivación directa de un grafo H' a partir de un grafo de entrada H aplicando una producción $P = (l, r, T, F)$, $H \xrightarrow{P} H'$ se define localizando un subgrafo de H, l^{host}, isomórfico con l, y reemplazando l^{host} por r^{host}, un subgrafo isomórfico con r. Sea $H^{-l^{host}}$ el grafo que permanece después de borrar l^{host} de H, las aristas entre el subgrafo r^{host} y $H^{-l^{host}}$ son *la inmersión* de r^{host} en $H^{-l^{host}}$, y se definen por medio de T. En el trabajo presente usamos isomorfismo inducido. Un subgrafo inducido de un grafo debe incluir todas las arista locales del grafo base, es decir todas las aristas que conectan dos nodos en el subgrafo. Para una explicación más detallada de gramáticas de grafo ver [87, 38, 40].

4.6.2 Definición de una gramática de grafo para representar símbolos texturados

Dado un símbolo texturado X que consiste en una repetición regular de n primitivas, definimos una gramática de grafo de atributos etiquetado que lo represente, $G = (\Sigma, \Delta, \Omega, P, S)$, de la siguiente manera:

- Dado el conjunto de n primitivas diferentes que forman X, $N_X = \{N_1, \ldots, N_n\}$:
  - Para cada N_i definimos una etiqueta para nodos no-terminales E_i, $i = 1, \ldots, n$.
  - Para cada N_i definimos una etiqueta para nodos terminales e_i, $i = 1, \ldots, n$.
  - Definimos la etiqueta del nodo inicial S'.
  - Obtenemos así, $\Delta = \{E_1, \ldots, E_n, S'\}$ y $\Sigma = \{e_1, \ldots, e_n\}$.
  - Para cada $N_i \in N_X$ denotamos como $C_i = \{C_1^i, \ldots, C_{m_i}^i\}$, $\forall j = 1 \ldots m_i$, $C_j^i \in R_X$, el conjunto de los diferentes tipos de vecindades que la primitiva N_i tiene en sentido antihorario. Por ejemplo en el símbolo que aparece en la Fig. 4.6(a), si N_0 es el rectángulo del centro, $C_0 = \{a, b, c\}$.
  - Para cada forma $N_i \in N_X$ denotaremos como $F_i = \{F_1^i, \ldots, F_{n_i}^i\}$, $\forall j = 1 \ldots n_i, F_j^i \in N_X$, al conjunto de vecinos directos que N_i tiene en sentido antihorario, comenzando por el F_1^i tal que la etiqueta de la arista (N_i, F_1^i) es C_1^i. Denotamos por $\mathcal{E}_i = \{E_1^i, \ldots, E_{n_i}^i\}$ a las correspondientes etiquetas no terminales del conjunto F_i. Por ejemplo, en la misma figura presentada en Fig. 4.6(a), si 0 es el rectángulo del centro, y denotamos cada nodo vecino con el número natural que tiene en su esquina superior, tenemos que $F_0 = \{1, \ldots, 6\}$.
- Dado el conjunto de los m tipos diferentes de vecindad que tiene X en su estructura, $R_X = \{R_1, \ldots, R_m\}$ definimos:
  - Para cada R_j definimos una etiqueta de arista terminal c_j.
  - Obtenemos así $\Omega = \{c_1, \ldots, c_m\}$.
- Las producciones de la gramática tienen sus nodos en l y r marcados con un número natural y un número natural con apóstrofe $'$ respectivamente. Estos números sirven para definir la inmersión. Así esta inmersión se representa con una lista de pares de números (q, q'), siendo q el número que hay sobre N_i, $N_i \in l$ y q' sobre N_j, $N_j \in r$. Los nodos de r que no tienen correspondencia en l se insertan de nuevo. Este tipo de inmersión es de conexión, véase Apartado 4.2.3, es decir determina cómo conectaremos los nodos del grafo r con los que queden del grafo de entrada una vez hayamos borrado el subgrafo l encontrado en él. Así el tipo de gramática de grafo será algorítmica, véase Apartado 4.2.3.

4.6.3 Inferencia de una gramática de grafo para representar símbolos texturados

Como hemos visto en el Apartado 4.5 de inferencia gramatical, la inferencia gramatical en casos prácticos no suele basarse en la teoría de lenguajes formales, sino en heurísticas y restricciones impuestas por las características del problema a resolver. En nuestro caso la inferencia gramatical que aplicamos se basa en el conocimiento que tenemos del tipo de grafos que vamos a generar. Éstos serán repetitivos con nodos de diversas clases según el número de primitivas que formen el símbolo, y con aristas etiquetadas según el tipo de vecindades que tenga cada nodo. También debemos notar que nuestra gramática será de grafo dado que lo que queremos generar y analizar son grafos, y además será dependiente de contexto ya que necesitamos saber que parte de los vecinos de un nodo dado se han analizado ya y cuales no para poder continuar con su análisis.

Dado un símbolo X con su conjunto de primitivas N_X y su conjunto de vecindades R_X. Tenemos para cada primitiva $N_i \in N_X$ un conjunto con sus diferentes tipos de vecindad en sentido antihorario $C_i = \{C_1^i, \ldots, C_{m_i}^i\}$, $\forall j = 1 \ldots m_i, C_j^i \in R_X$. Siendo $F_i = \{F_1^i, \ldots, F_{n_i}^i\}$, $\forall j = 1 \ldots n_i, F_j^i \in N_X$, el conjunto de vecinos directos que N_i tiene en sentido antihorario, comenzando por el F_1^i tal que la etiqueta de la arista (N_i, F_1^i) es c_k siendo $C_1^i = R_k$, $R_k \in R_X$. Inferiremos la gramática que lo representa de la siguiente manera:

1. Para cada $N_i \in N_X$ definimos una producción $P = (l, r, T, A)$ donde:

   - l está formada por el nodo inicial S'.
   - r es el subgrafo con un nodo n etiquetado como e_i, y con todos los ciclos cerrados que empiezan en N_i.
   - Los nodos vecinos de n se etiquetan con las etiquetas de nodo no-terminal siguientes: $\{E_1^i, \ldots, E_{m_i}^i\}$ en el sentido antihorario.
   - Se etiquetan las aristas entre todas ellas con las correspondientes etiquetas de arista terminal.
   - $T = \{(1, 1')\}$ apareciendo sobre S' un 1 y sobre el nodo n del subgrafo r un $1'$.

2. Para cada nodo no-terminal podemos tener uno o más nodos terminales como vecinos. Cada nodo terminal tiene todos sus vecinos, por lo menos etiquetados como no-terminales. Entonces generamos para cada forma N_i y cada número de nodos terminales que N_i puede tener, un conjunto de producciones, es decir generamos un conjunto de producciones cuando N_i tiene un nodo terminal como vecino, cuando tiene dos y así sucesivamente hasta que generamos una regla en la que tiene todos sus n_i vecinos como terminales. Es decir para $j = 1$ hasta n_i. Cada producción de cada uno de estos conjuntos de producciones se define cogiendo el primer vecino terminal que sigue una de las posibles vecindades, es decir para $k = 1$ hasta m_i. Debemos observar que cada tipo de vecindad se

representa a sí misma y a ella con una rotación de π radianes. Así dada una forma N_i, el número j de vecinos terminales que consideramos en esa producción, y la posición k en F_i que tiene el primer vecino terminal, definimos una producción que tiene como l un grafo que tiene un nodo n etiquetado como E_i, y con un número 1 sobre él, y sus j primeros vecinos en F_i comenzando por F_k^i etiquetados con sus etiquetas de nodos terminales correspondientes. Cuando todos los nodos terminales han sido insertados, se han de insertar los vecinos que éstos tienen en común con el nodo n que no estén todavía en l. Una vez hecho esto r se definirá copiando l y cambiando la etiqueta de n por e_i, es decir convirtiendo n en un nodo terminal, y cambiando cada número q sobre un nodo de l por un número q' sobre el nodo de r correspondiente. Después se añaden el resto de vecinos que n puede llegar a tener, etiquetándolos con sus etiquetas de nodo no-terminal correspondientes.

3. Para cada N_i se añade una regla de terminación, siendo l un nodo etiquetado con E_i y con un número 1 sobre él. r es un nodo etiquetado con e_i y un número $1'$ sobre él. Es decir son reglas que sustituyen un nodo no-terminal por su correspondiente terminal.

En las figuras Fig. 4.7, Fig. 4.8, y Fig. 4.9 vemos algunos ejemplos de gramáticas inferidas de esta manera. La gramática que aparece en la Fig. 4.7 corresponde al símbolo texturado de la Fig. 4.6(a). La gramática que aparece en la Fig. 4.8 corresponde a los símbolos texturados de las Fig. 4.6(c), y Fig. 4.6(d). La gramática que aparece en Fig. 4.9 corresponde al símbolo texturado de la Fig. 4.6(b).

4.6.4 Ampliación de la gramática de grafo con producciones de error

Como hemos dicho a lo largo de este capítulo, en las aplicaciones prácticas tanto de definición de una gramática como de inferencia de la misma, hemos de tener en cuenta las posibles deformaciones que puede tener un patrón. En nuestro caso esas deformaciones pueden ser debidas a diversas causas. Primero al pasar de un documento gráfico a un grafo que lo represente, hemos de realizar un escaneado de la imagen y una vectorización de la misma que nos dé los segmentos de los que está formada. Ambos procesos pueden introducir distorsiones en el símbolo. Además dada la naturaleza finita del símbolo podemos encontrarnos con otro tipo de distorsiones, como fragmentación de las primitivas que aparecen en el borde del mismo. Así tanto si es debido al proceso de obtención del grafo, escaneado o vectorización, como si es debido a la naturaleza finita del símbolo podemos encontrar cuatro tipos de distorsiones que se presentan en la Fig. 4.12:

1. Distorsiones en las primitivas que forman la textura, ver Fig. 4.12(a).

2. Distorsiones en las reglas de emplazamiento de las primitivas que forman la textura, ver Fig. 4.12(b).

3. Fusión de dos o más elementos que forman la textura, ver Fig. 4.12(c).

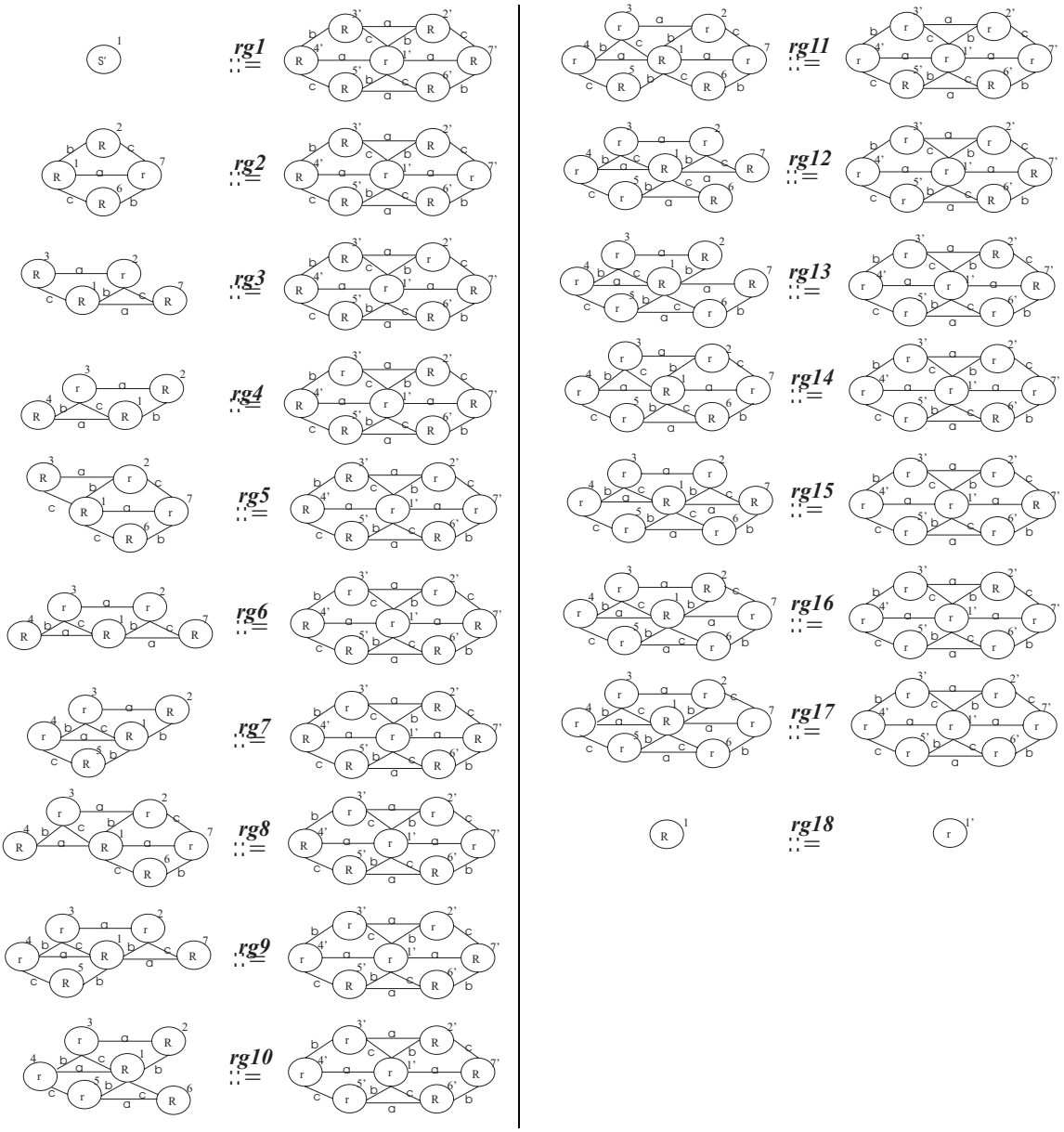


Figura 4.7: Gramática de grafo que representa el símbolo texturado de la Fig.4.6(a).

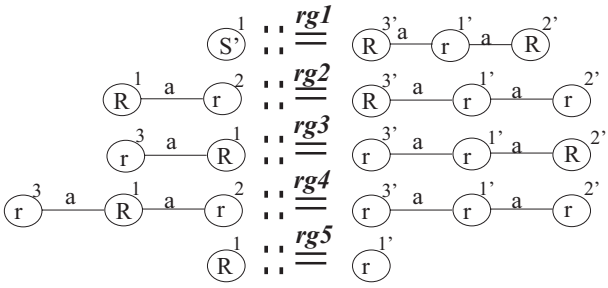


Figura 4.8: Gramática de grafo que representa el símbolo texturado de la Fig.4.6(c) y de la Fig.4.6(d).

4. Oclusión parcial de primitivas que forman la textura, debido a la naturaleza finita del área texturada, ver Fig. 4.12(d).

Como hemos visto en el Apartado 4.3 existen diversas formas para poder reconocer patrones con ciertas distorsiones o errores por medio de una gramática. En nuestro caso necesitamos corregir ciertos errores tipificados *a priori* como podrían ser: las formas fusionadas o la oclusión de formas. Para ellos podríamos simplemente crear unas reglas de corrección de error que tuvieran en cuenta estos casos, añadiéndoles un cierto coste a la hora de reconocer patrones que estuvieran distorsionados. Pero también tenemos otro conjunto de errores que no son tipificables como son las distorsiones en la forma del textón que forma la textura o en su regla de emplazamiento. Para estos errores necesitaremos que sea el algoritmo de análisis sintáctico el que determine si una forma corresponde con la esperada o no, y lo mismo pasa con las reglas de emplazamiento de las mismas. Por todo ello necesitaremos una gramática que por un lado tenga producciones de error para las distorsiones tipificadas pero por otro tenga una analizador sintáctico con corrección de error para las no tipificadas.

Así, ahora para modelizar un símbolo texturado X que consista en una repetición regular de n primitivas diferentes, $N_X = \{N_1, \ldots, N_n\}$, y m tipos diferentes de vecindades $R_X = \{R_1, \ldots, R_m\}$, el proceso de inferencia será como sigue:

1. Para cada primitiva N_i que forme X, se definirá una jerarquía de etiquetas de nodos no terminales E_i, y tres etiquetas de nodos terminales e_i, ce_i, de_i, que representen la primitiva, la primitiva cortada en el borde y más de una primitiva fusionadas respectivamente. Después se definirá la etiqueta del nodo inicial S', obteniendo así $\Delta = \{E_1, \ldots, E_n, S'\}$ y $\Sigma = \{e_1, ce_1, de_1 \ldots, e_n, ce_n, de_n\}$. Para cada jerarquía L_i definimos dos etiquetas derivadas, IE_i y RE_i, que representan un nodo insertado y un nodo real, respectivamente.

2. Se define una etiqueta de arista terminal c_i para cada tipo de vecindad R_i, obteniendo así $\Omega = \{c_1, \ldots, c_m\}$. Para cada primitiva $N_i \in N_X$ denotamos como $C_i = \{C_1^i, \ldots, C_{m_i}^i\}$, $\forall j = 1 \ldots m_i, C_j^i \in R_X$, al conjunto de los diferentes tipos de vecindades que tiene N_i en sentido horario, por ejemplo para el símbolo en Fig. 4.6(a), si N_i es el rectángulo en el centro, $C_i = \{a, b, c\}$.

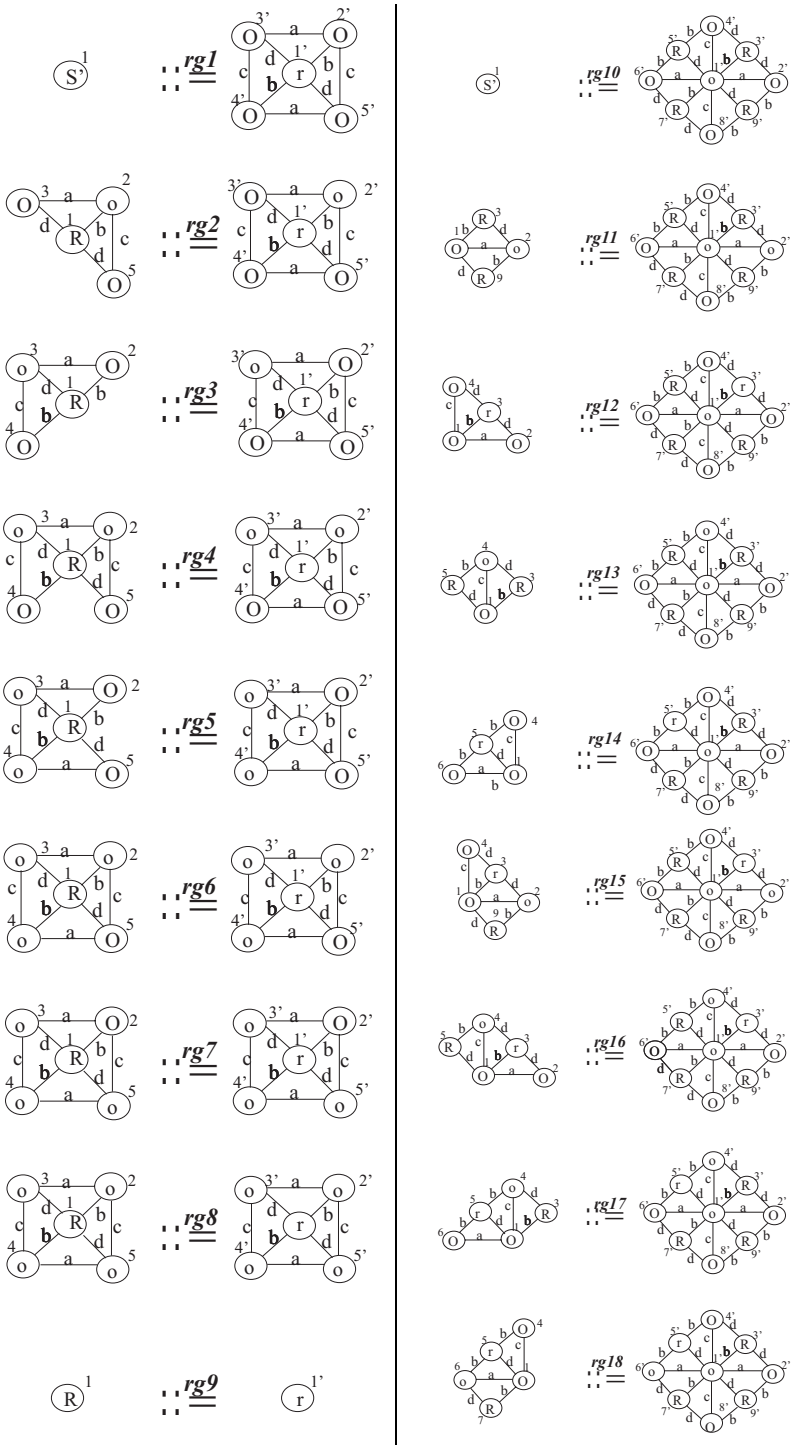


Figura 4.9: Gramática de grafo que representa el símbolo texturado de la Fig.4.6(b). Parte (I)

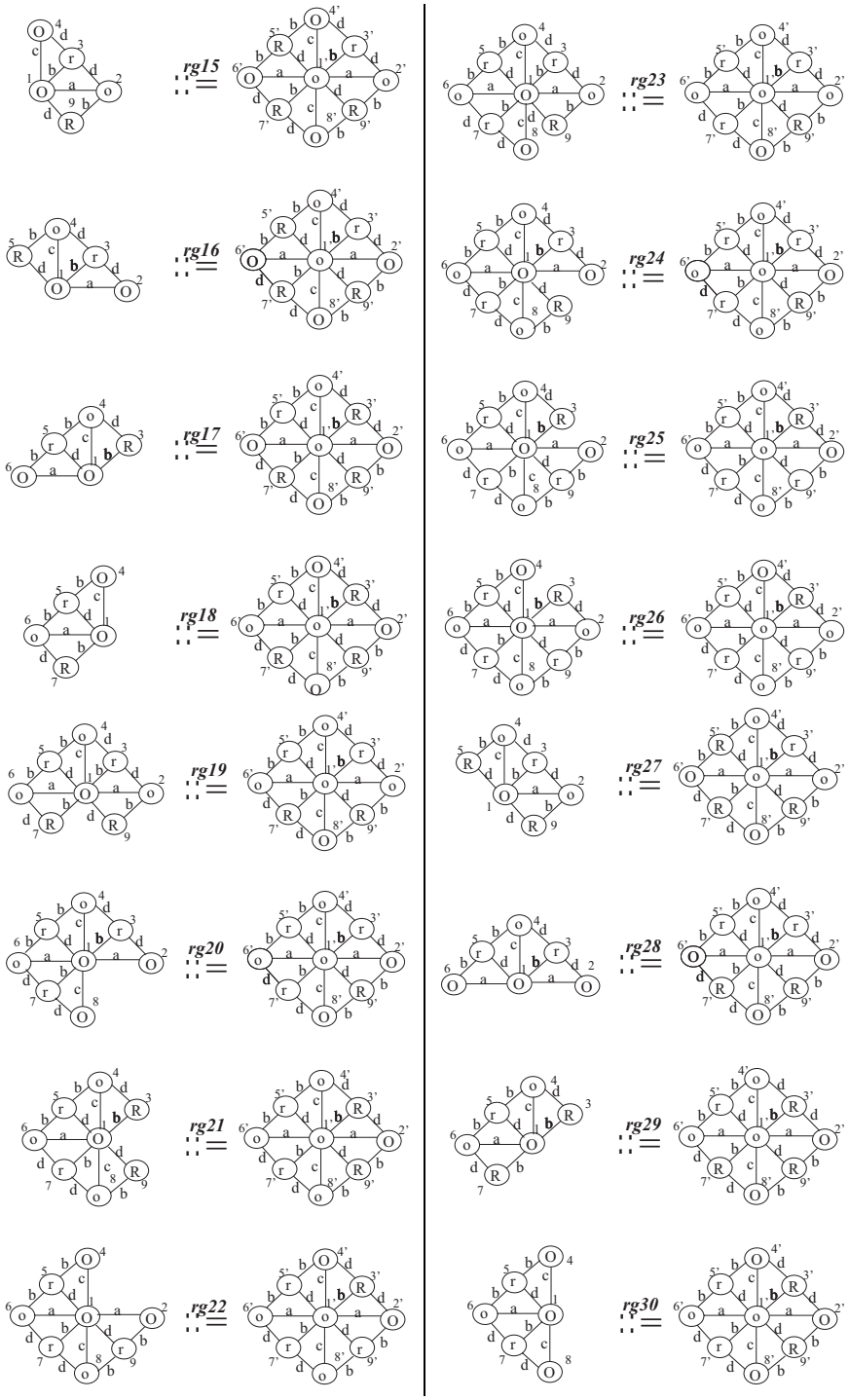


Figura 4.10: Gramática de grafo que representa el símbolo texturado de la Fig.4.6(b). Parte (II)

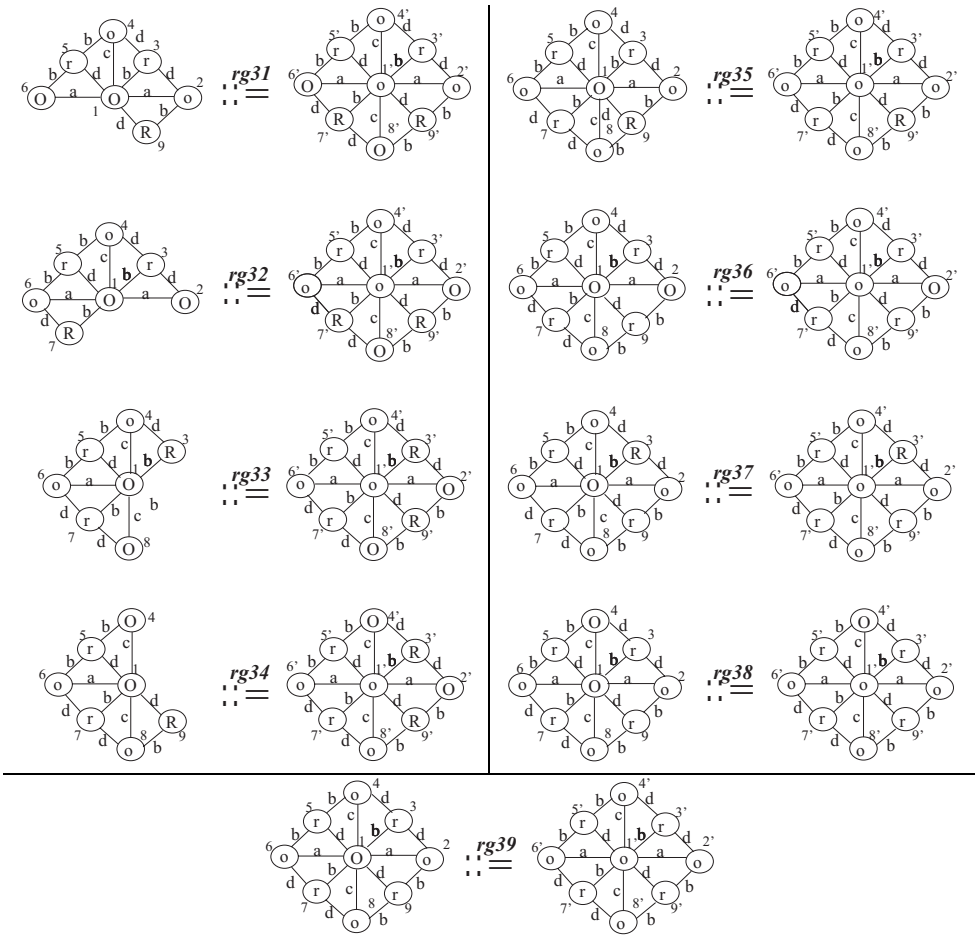


Figura 4.11: Gramática de grafo que representa el símbolo texturado de la Fig.4.6(b). Parte (III)

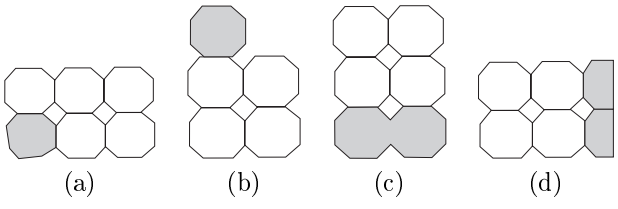


Figura 4.12: Posibles distorsiones en la obtención del grafo. (a) Forma distorsionada. (b) Reglas de emplazamiento distorsionadas. (c) Formas fusionadas. (d) Oclusión de formas.

3. Para cada primitiva $N_i \in N_X$ denotamos como $F_i = \{F_1^i, \ldots, F_{n_i}^i\}$, $\forall j = 1 \ldots n_i, F_j^i \in N_X$, al conjunto de vecinos que tiene N_i en sentido horario, empezando por el F_1^i tal que la etiqueta de la arista (N_i, F_1^i) sea R_1^i, por ejemplo en el mismo símbolo que aparece en Fig. 4.6(a), siendo N_i el rectángulo del centro, y denotando cada nodo vecino con el número natural que el rectángulo tiene en su borde, es decir $N_1, \ldots, N_6$, $F_i = \{N_1, \ldots, N_6\}$. Entonces las producciones se definirían de la siguiente forma:

   (a) Para cada $N_i \in N_X$ definimos una producción que tenga por parte izquierda el nodo inicial S', y por parte derecha un subgrafo con un nodo n etiquetado como e_i, y con todos los ciclos cerrados que empiezan en N_i, etiquetando los nodos vecinos de n con las siguientes etiquetas de nodo no-terminal $\{E_1^i, \ldots, E_{m_i}^i\}$ en el sentido horario, y con las aristas entre ellas con sus correspondientes etiquetas terminales.

   (b) Cada nodo no terminal puede tener uno o más nodos terminales como vecinos. Cada nodo terminal tiene todos sus vecinos, etiquetados, por lo menos como no terminales. Entonces para cada forma N_i, se genera un conjunto de producciones para cada número de nodos terminales que pueda tener N_i, es decir se genera un conjunto de producciones cuando N_i tiene un nodo terminal, cuando tiene dos y así hasta que tenga n_i, y cada producción dentro de ese conjunto se define cogiendo el primer vecino terminal siguiendo una de las posibles vecindades.

   (c) Los nodos insertados nos permiten terminar la textura y corregir los errores de las formas que aparecen sólo parcialmente porque están en la frontera de la textura o de las formas que están fusionadas debido a distorsiones en el proceso de adquisición. Para estos nodos insertados, se añade una regla que permite: sustituirlos por λ, por el símbolo cortado terminal correspondiente ce_i, o por el símbolo fusionado terminal correspondiente de_i.

La Figura 4.13 nos muestra un ejemplo de una gramática de grafo que representa la textura de la Fig. 4.6(b). Hay dos tipos de primitivas formando la textura: el rectángulo y el octógono. Para el octógono, se definen tres etiquetas terminales: *o*, *co*, *do*, que representan el octógono sin deformaciones, el octógono cortado, y el conjunto de octógonos fusionados en una sola forma, y una jerarquía de etiquetas no terminales, O, que tiene como etiquetas derivadas, *Ro* que representa un nodo existente, y *Io*, que representa uno insertado. Para el cuadrado, se definen otras tres etiquetas de nodos terminales, *r*, *cr*, *dr*, que representan el cuadrado normal, el cortado y el fusionado respectivamente, y una jerarquía de etiquetas no terminales, R, que tiene como etiquetas derivadas, *Rr* que representa un nodo existente, y *Ir*, que representa uno insertado. Para cada producción P, l y r son grafos cuyos nodos tienen un número q o un número con $'$ q', este número representa la regla de inmersión, es decir el nodo de la parte izquierda con un número i corresponde con el nodo de la parte derecha con un número i'. Las reglas 1 y 10 son las reglas de inicio para el cuadrado y el octógono respectivamente, mientras que las reglas 9 y 40 son las de

final y corrección de errores para el cuadrado y el octógono respectivamente. Debemos observar que cada vez que la etiqueta O aparece en una producción nos podemos estar refiriendo a cualquiera de sus dos etiquetas derivadas Io y Ro, y lo mismo pasa con la etiqueta R, respecto a sus etiquetas derivada Ir y Rr. Sin embargo, sólo se puede reescribir un nodo no terminal por un terminal cuando es un nodo real, es decir, cuando corresponde a un Ro o un Rr, siendo las únicas excepciones las reglas para corregir errores. Deberíamos remarcar que cada regla se representa a ella misma y a ella rotada ϕ radianes, y que las reglas con parámetros $e1$ y $e2$ están representando cuatro tipos de reglas: las que no tienen los nodos y aristas parametrizados con estas $e1$ y $e2$, las que tienen los nodos parametrizados con $e1$, las que tienen los nodos parametrizados con $e2$, y las que los tienen todos.

4.6.5 Reconocimiento de símbolos texturados

Una vez tengamos el conjunto de n gramáticas G_i, $i = 1, \ldots, n$, que representen los símbolos texturados X_i, $i = 1, \ldots, n$ a reconocer. Dado un grafo de entrada H que represente un dibujo de arquitectura, el proceso de análisis sintáctico sobre H, para reconocer el símbolo texturado, consistirá en ir recorriendo los nodos de H que cumplan las reglas de forma y emplazamiento que definen a X_j, $j \in \{1, \ldots, n\}$. Es decir iremos agrupando, y marcando como parte del símbolo, los nodos que cumplan estas características y desechando y marcando como no-símbolo los que no las cumplan. De esta forma, al final del proceso, tendremos uno o varios conjuntos d e nodos conectados por sus vecindades que formaran el símbolo texturado, o varias ocurrencias del mismo, o todos los nodos marcados como rechazados lo que nos indicará que el símbolo X_j no aparece en H. Si sólo queremos reconocer un símbolo el proceso habrá terminado, sino el proceso se repetirá para cada uno de los símbolos X_i, $i = 1, \ldots, n$.

Así el algoritmo del proceso formal de reconocimiento de un símbolo texturado quedaría de la siguiente forma:

- Primero los nodos en H se toman de uno en uno hasta que una regla de inicio pueda aplicarse sobre uno de ellos. Llamémosle n a este nodo seleccionado. Esto quiere decir que la forma representada por n es similar a la forma de la primitiva que nos esperábamos en la parte izquierda de la regla de inicio, y que existe un isomorfismo de subgrafos entre un subgrafo alrededor de n y el grafo h_r de esa regla de inicio. La similitud entre las formas que están representadas por los nodos se calcula por medio de la distancia de cadenas, como se describe en el capítulo 2.

- En este proceso, n se marca como un elemento que forma el símbolo texturado. El conjunto de nodos y aristas de H alrededor de n que tiene una equivalencia con la parte derecha de la regla gramatical se etiqueta como indique la regla, siendo los nodos etiquetados como no terminales insertados en una lista, Lnt, para ser analizados por el analizador sintáctico, mientras que los nodos y aristas que no se han encontrado se etiquetan como nodos insertados y se añaden a otra lista, Li, para ser analizados durante una fase de postproceso, para encontrar posibles errores y terminar el análisis del símbolo texturado. Estos nodos insertados están apuntando a nodos existentes en H, los cuales están en una posición

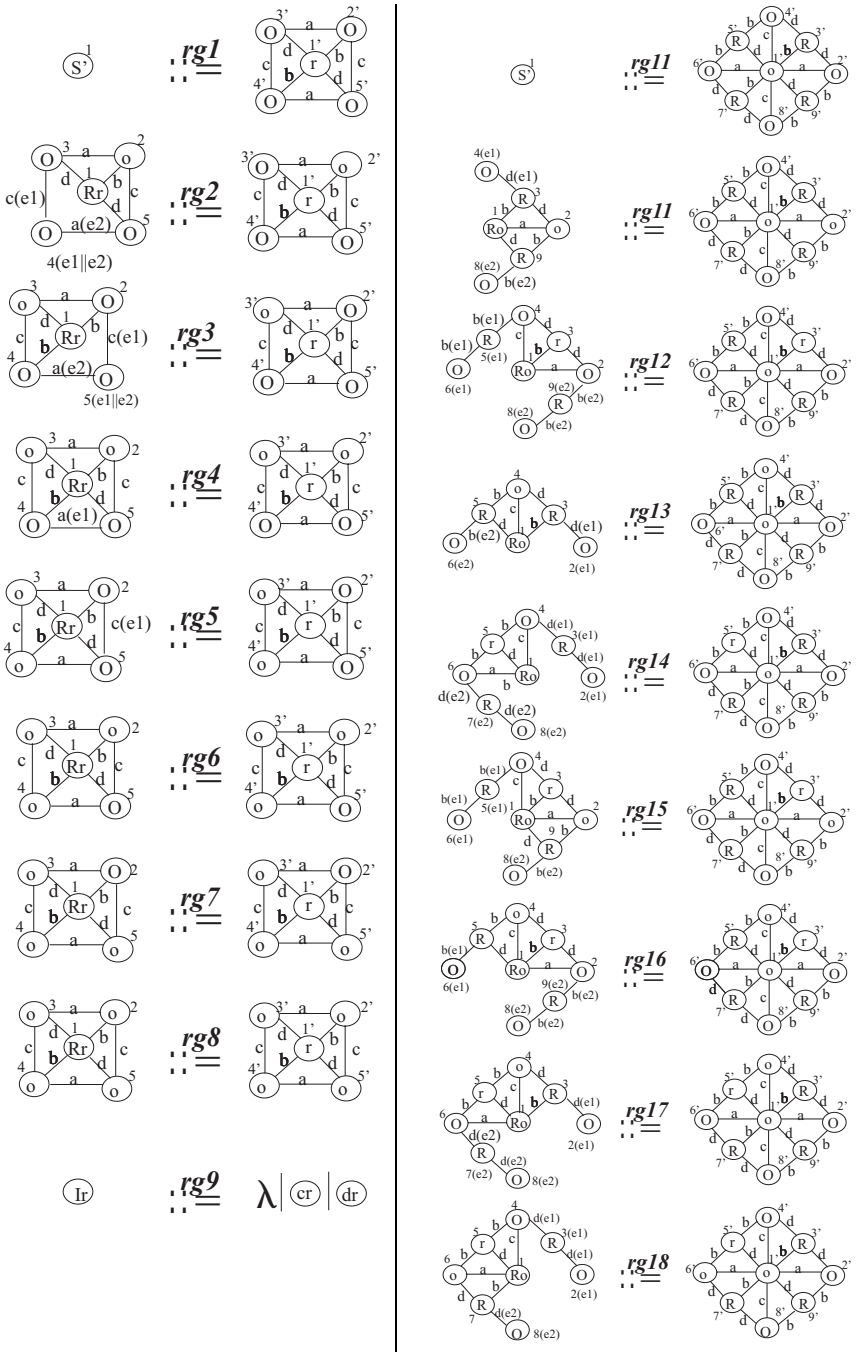


Figura 4.13: Gramática de grafo que representa el símbolo texturado de la Fig. 4.6(b). Parte I

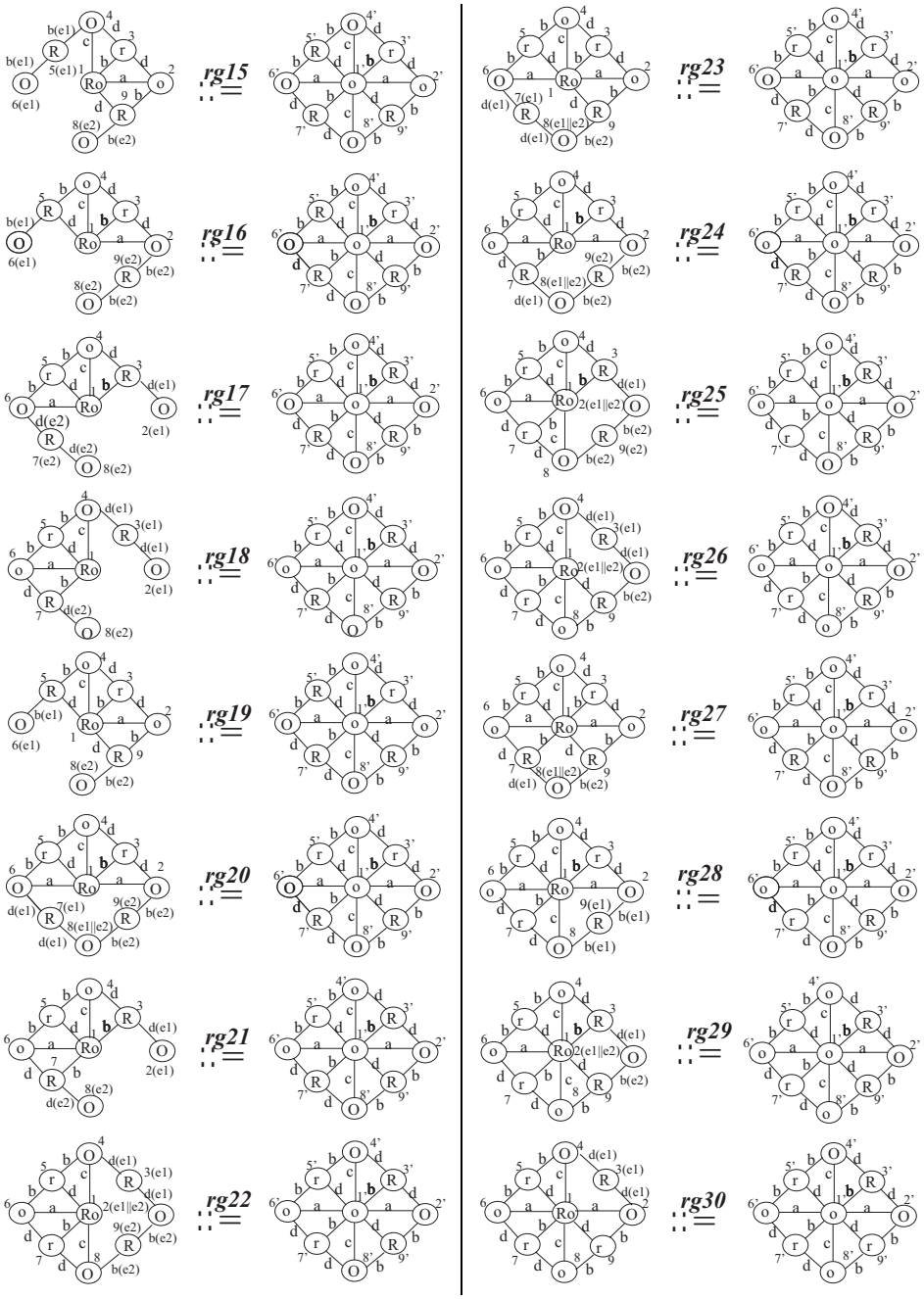


Figura 4.14: Gramática de grafo que representa el símbolo texturado de la Fig. 4.6(b). Parte II

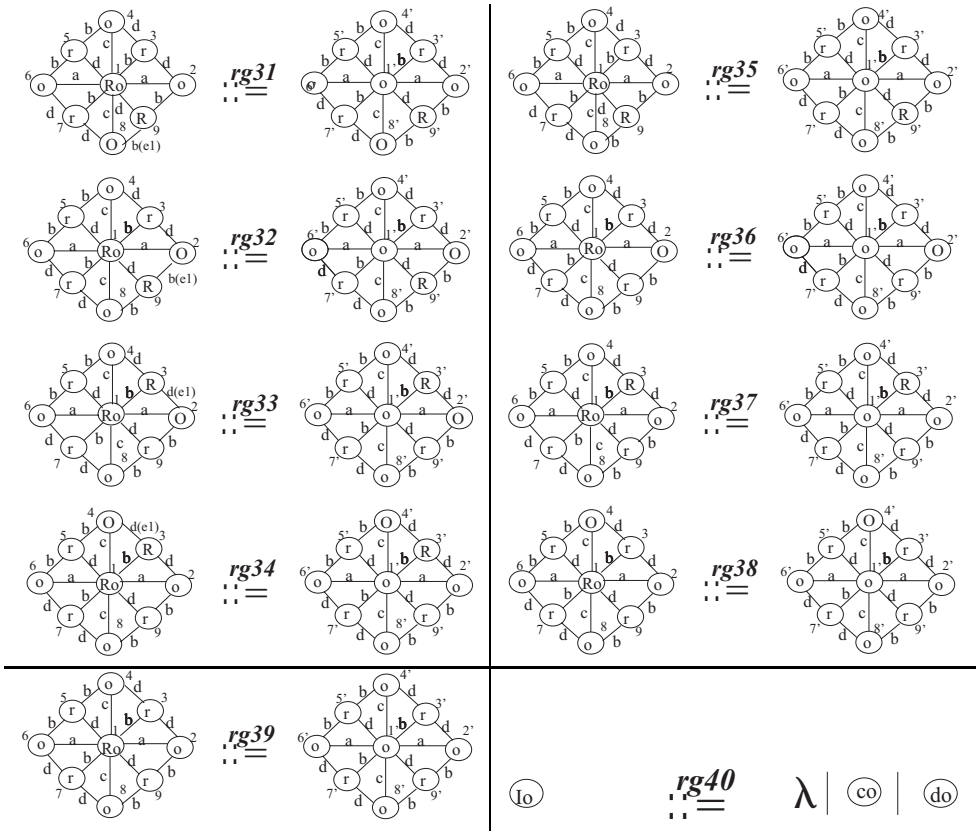


Figura 4.15: Gramática de grafo que representa el símbolo texturado de la Fig. 4.6(b). Parte III

relativa con respecto a n similar a los nodos esperados correspondientes de el h_r de la regla gramatical.

- Así para cada nodo no terminal n en Lnt, la regla que se aplicará cada vez será seleccionada directamente contando el número de nodos vecinos terminales que tiene y en que posición empiezan. Luego la regla se aplica de la misma manera que en el paso previo, usando un isomorfismo de subgrafos e insertando en Lnt y Li los nodos no terminales y los insertados respectivamente para ser analizados posteriormente. El nodo n se marca como un nodo que forma parte del símbolo texturado.

- Una vez han sido analizados todos los nodos no terminales de Lnt, se tiene un conjunto de nodos en H marcados como parte del símbolo texturado, y alrededor de ellos, los nodos marcados como insertados.

- Para cada nodo n insertado en Li, se aplica una regla de corrección de error o de finalización. Las reglas de finalización borran los nodos insertados junto con sus aristas, mientras que las de corrección de errores verifican si existe una forma cortada o una forma unida a varias en la posición del nodo insertado. Cada vez que se aplica una regla de corrección de error, se le debe asociar un coste, que cuantifique la distorsión que tiene el símbolo texturado respecto al modelo. Una vez que ese coste acumulado alcanza un cierto límite, no es posible realizar más corrección de errores, y el resto de modos insertados serán borrados y no considerados como parte del símbolo texturado.

En la Fig. 4.16, se muestra un ejemplo del proceso de análisis sintáctico de un grafo respecto a una gramática de grafo que representa un símbolo. Dado el grafo H en la Fig. 4.16(b) que representa la imagen de la Fig. 4.16(a) se realiza el análisis sintáctico del mismo siguiendo la gramática que se presenta en la Fig. 4.13. En la Fig. 4.16(c) se coge el primer nodo y se compara con las dos primitivas posibles que forman el símbolo, y se obtiene que corresponde al octógono. Luego se aplica la regla de inicio del octógono, es decir la regla número 10 en la Fig. 4.13, obteniendo el grafo que aparece en la Fig. 4.16(c), donde los nodos etiquetados con Ir o Io son los nodos insertados, el nodo con una o blanca es el que se ha reescrito como nodo terminal, y los nodos con R y O son los que se han reescrito como nodos no terminales. Luego todos los nodos no terminales se ponen en una lista Lnt y los insertados en otra Lin separada. Se coge el primer no terminal de la lista Lnt y se aplica la regla 11 en su forma simétrica y con sus parámetros $e1$ y $e2$ a falso, modificándose el grafo como aparece en la Fig. 4.16(e), donde los nodos que ya han sido analizados y considerados como parte del símbolo aparecen en color gris. De la misma manera, se aplican las reglas marcadas en las Fig. 4.16(e)–(m), donde los parámetros T o F denotan si los nodos etiquetados con $e1$ y $e2$ existen o no respectivamente, y $Sym.$ denota si la regla se ha aplicado en su version simétrica o no. En la Fig. 4.16(f) se aplica la regla 13($false, false$) de la Fig. 4.13, con el grafo resultante de la Fig. 4.16(f); Debemos notar que esta forma tiene alguna distorsión, pero puede ser reconocida como el octógono porque esas distorsiones están por debajo de un cierto límite. Una vez que ha sido analizada toda la lista de nodos no terminales, estaremos en el punto de la Fig. 4.16(n), donde todos los nodos que

han sido reconocidos como pertenecientes al símbolo que representa la gramática de la Fig. 4.13 están en color gris, y todos los nodos insertados aparecen alrededor de ellos. Luego se aplican las reglas gramaticales 9 y 40, para borrar los nodos insertados que no existen obteniendo el grafo que aparece en la Fig. 4.16(o). A partir de este punto se aplica la regla 40 para corregir errores, seleccionando las formas cortadas y las fusionadas, representadas por los nodos terminales *oc* y *os*, respectivamente. Al final, todo el grafo se reconoce como el símbolo texturado representado por la gramática.

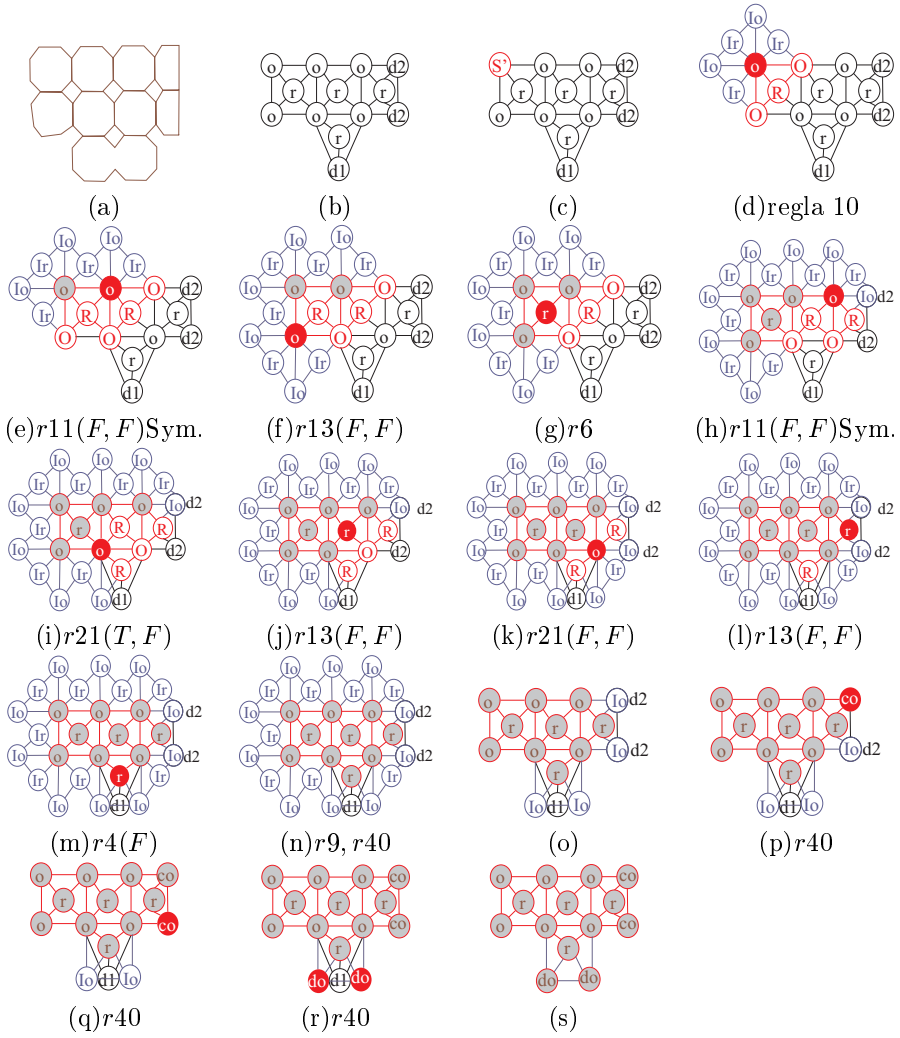


Figura 4.16: Proceso de reconocimiento sintáctico de un símbolo texturado.

4.6.6 Conclusiones

Uno de los objetivos de este trabajo es la modelización y posterior reconocimiento de símbolos texturados en documentos gráficos. En este capítulo hemos presentado la herramienta que hemos escogido para tal efecto, que es una gramática de grafo con atributos. Una primera definición de la gramática nos permite modelizar símbolos texturados infinitos, cuyo parser tolera ciertas distorsiones en las formas que componen los texels del símbolo y en sus vecindades. Esta definición vemos que no nos permite reconocer texturas reales que son finitas, y en las que aparecen errores mayores como los producidos por una fragmentación, fusión o división en los texels de la textura, así como la presencia de agujeros en la misma. Para poder reconocer texturas con este tipo de distorsiones definimos una gramática con corrección de errores que permite al parser generar nodos insertados que se corresponden con posibles nodos del grafo original para así poder realizar una comprobación de si éstos corresponden a texels partidos, fusionados o divididos, o simplemente en ese espacio existe un agujero en la textura. Este proceso de comprobación del error debería tener un coste asociado que controlara que la distorsión es tal y no corresponde en realidad a otro símbolo texturado. Además necesitamos una función que determine si las formas asociadas a los nodos insertados en el proceso de parser corresponden realmente a texels partidos o fusionados o son otro tipo de elementos.