UNIVERSIDAD AUTÓNOMA METROPOLITANA IZTAPALAPA
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

DETECCIÓN DE COMPLEJOS QRS NORMALES EN ELECTROCARDIOGRAFÍA AMBULATORIA UTILIZANDO LA INFERENCIA GRAMATICAL

TESIS PARA OBTENER EL GRADO DE
MAESTRÍA EN INGENIERÍA BIOMÉDICA

OSCAR YAÑEZ SUÁREZ

ASESORA: M. EN I. SONIA CHARLESTON VILLALOBOS

México, D. F., enero 1993

CONTENIDO

INTRODUCCION

Cuando un paciente le refiere a su médico síntomas eventuales de alteraciones de la presión sanguínea, desmayos, taquicardia y otros no específicos difíciles de diagnosticar en una sesión de consulta común, el médico se topa con un problema importante de diagnóstico, ya que el poder asociar estos síntomas con diversos padecimientos se requiere de la aplicación de una estrategia inteligente de identificación de la enfermedad.

En el caso de sospecha de alteraciones cardiológicas, es deseable obtener la mayor información posible del comportamiento de la señal electrocardiográfica (ECG) del paciente. La electrocardiografía ambulatoria, introducida por N.J. Holter en 1961 resuelve el problema de captura y análisis de grandes registros de señal electrocardiográfica [16,18]. La técnica consiste en el registro continuo por 24 horas del ECG del sujeto bajo estudio durante el desarrollo de su actividad diaria, para que ésta sea analizada posteriormente en clínica [10,11,20].

El registro de la señal de ECG por periodos largos puede hacerse convenientemente con una unidad portátil de detección y registro, que se pueda llevar cómodamente durante la realización de las actividades cotidianas (la

descripción de la unidad portátil relacionada con el proyecto motivo de esta tésis se encuentra en Jiménez et al.[24]). La señal así registrada puede analizarse posteriormente, en forma manual, semi-automática o automática [3,25,31]. La disponibilidad de esta información le permite al médico tomar una decisión de los pasos a seguir en el tratamiento de su paciente.

Los sistemas portátiles pueden ser del tipo analógico, que operan por medio de la grabación de la señal en cinta de audio, o también digitales, los cuales discretizan la señal y la almacenan en memoria de semiconductor. Existe también la opción de almacenamiento digital en cinta de audio. Hasta 1992, todos los sistemas Holter comerciales eran de almacenamiento en cinta; en ese año se comercializó una unidad totalmente digital. Esto ha sido así debido a los múltiples problemas tecnológicos que se imponen al desarrollar una unidad de este tipo [17,24,40].

Un problema presente en el monitoreo ambulatorio clásico del ECG es que del total de complejos registrados sólo algunos aportan información útil, de modo que se almacena gran cantidad de información redundante. Otro conflicto es el propio medio de registro, del que se conocen bien sus limitaciones prácticas. Sustituir la cinta de audio por memoria de semiconductor implicaría la necesidad de establecer una estrategia de compresión de la información

pues los registros crudos requerirían de algunas decenas de megabytes de almacenamiento, lo cual resulta impráctico. Esto hace pensar que un sistema que realice un registro en forma dinámica, comprimiendo la señal y seleccionando la información que almacena, resultaría más favorable [17].

La compresión de datos de electrocardiografía ha sido un tema que ha acaparado la atención de los investigadores por muchos años. La necesidad de la transmisión de datos de ECG por vía telefónica a los hospitales fue quizá el detonante de esta área de desarrollo tecnológico. Desde 1968, el algoritmo AZTEC [8], diseñado con este propósito ha sido uno de los algoritmos de procesamiento del ECG más utilizado en el mundo. Aún cuando los algoritmos de compresión han proliferado por decenas [22], hay indicadores que muestran la permanencia de AZTEC como el algoritmo de elección en varias aplicaciones [44].

Un sistema de registro dinámico debe incluir algoritmos de análisis del ECG que permitan seleccionar trozos de señal para ser almacenados o desechados. La existencia de estos sistemas se basa en el diseño adecuado de un detector del complejo QRS, o específicamente de la onda R [7,12,32,40]. Estos detectores suelen ser sistemas dedicados independientes, ya sean analógicos o digitales. Los sistemas analógicos son eficientes pues generan información en línea, pero provocan problemas de diseño por el exceso de

componentes electrónicos que requieren. Los sistemas digitales, por el contrario, pueden realizar esta tarea con menos componentes, pues los mismos circuitos pueden ser usados para detectar la onda R y analizar en conjunto a la señal. Sin embargo, entre las técnicas digitales utilizadas actualmente muy pocas pueden operar en línea [23,32,43].

La detección digital de complejos QRS se ha abordado desde diversas perspectivas, como son:

- determinación de máximos y mínimos locales [9]
- filtros pareados (matched filters) y evaluación de funciones de correlación [28]
- predicción lineal [29,30]
- filtrado adaptable [39]
- análisis sintáctico [19,34,42]

Friesen y cols.[12] y Pahlm y Sornmo [32] han realizado excelentes revisiones sobre los métodos de detección de los complejos QRS. En estas revisiones se ha otorgado especial importancia a los métodos que se enmarcan en la teoría de los sistemas lineales, dejando a un lado los relativos al análisis sintáctico. Sin embargo, la complejidad de éstos, comparada con la de los primeros es notablemente menor, desde el punto de vista de las demandas de cálculo que se imponen al sistema donde son implantados. Para una

plataforma digital sencilla [24] los algoritmos sintácticos resultan una buena opción, como se discutirá a lo largo del presente trabajo.

CAPITULO 1

ANALISIS ESTRUCTURAL SINTACTICO APLICADO A LA ELECTROCARDIOGRAFIA

Al abordar problemas de reconocimiento y clasificación de patrones, como el que ha sido motivo del presente trabajo, se tiene la disyuntiva de elegir entre dos grandes esquemas metodológicos, el análisis de distancia o la caracterización morfológica [15]. De éstos, el primero puede ser considerado como el método clásico de trabajo, mientras que el segundo ha tenido, comparativamente al primero, pocos espacios de aplicación. Sin embargo en muchos casos ambas estrategias pueden resultar complementarias.

Desde el punto de vista del análisis de distancia es necesario relacionar a cada patrón con una colección de medidas que lo identifiquen de la manera más completa posible. Al mismo tiempo, cada clase de patrones del total en las que se ha dividido el problema se debe relacionar con un conjunto "tipo" de estas mismas medidas. Estos conjuntos de medidas pueden ser representados en forma lógica como vectores (vectores de características), lo cual permite establecer una regla de decisión relativa a la pertenencia o

no de un patrón dado a una cierta clase basada en la evaluación de una medida de distancia entre el vector de características del patrón problema y el correspondiente vector de la clase con la que se busca la asociación. De esta forma, para resolver el problema debe establecerse adecuadamente el espacio de coordenadas (medidas incluidas en el vector de características), así como la naturaleza de las fronteras espaciales que engloban a cada clase de patrones (reglas o funciones de decisión).

En la caracterización morfológica se parte de la aseveración de que cada patrón problema y cada clase de patrones tienen una estructura inherente que los identifica, y que por consiguiente el problema de reconocimiento o clasificación de un patrón puede establecerse como la comparación de esas estructuras. Estas tienen una composición propia, formada por sub-estructuras organizadas en una forma determinada. Para poder describir analíticamente esta organización se recurre a una representación de los patrones mediante unidades mínimas, llamadas **primitivas**, cuyo ensamble ordenado dé origen a cada clase de patrones.

Estas primitivas deben ser usadas para describir todos los patrones que se desean clasificar. Si el orden de las primitivas del patrón problema es igual al que describe a una cierta clase, entonces el patrón se asocia con ésta.

Desde el punto de vista práctico, la alternativa morfológica impone los requisitos de seleccionar las primitivas de acuerdo al problema particular y de utilizar un método estandarizado de representación estructural.

Es interesante resaltar el paralelismo existente entre las metodologías descritas; en ambas se requiere resolver un problema de representación (vector de características o definición de primitivas) y un problema de análisis (medidas de distancia o descripción estructural). Mientras que el problema de análisis está razonablemente resuelto (espacios euclidianos, medidas estadísticas, lenguajes formales), el problema de representación es básicamente uno de creatividad, y de su adecuada solución depende en buena medida el éxito de la clasificación de los patrones.

El ECG, por tener una forma de onda notoriamente distintiva ha resultado buen candidato para ser representado estructuralmente en primitivas; de hecho, el análisis tradicional que del registro de electrocardiografía realiza un médico parte del reconocimiento de sub-ondas dentro del ciclo completo R-R. Sin saberlo concientemente, un médico realiza la caracterización morfológica del ECG con una metodología idéntica a la descrita anteriormente.

Con el fin de explicar sistemática y formalmente la caracterización morfológica de patrones es necesario recurrir a la teoría de los lenguajes formales [1,35], misma

que aparece como el sostén de la ciencia de la computación. Describir una estructura de control como "if...then...else" en el lenguaje de programación C, darle significado y probar su corrección sintáctica es justamente lo mismo que describir un ciclo de ECG como "onda P...complejo QRS...onda T", asociarle una interpretación fisiológica y comprobar la aparición de cada onda en formas y tiempos característicos. La verificación del orden y asociación de las primitivas de un patrón se fundamenta en el análisis estructural sintáctico [6].

Tal y como un lenguaje de programación tiene símbolos predefinidos con los cuales se construye un programa (tokens), un lenguaje que describe patrones tiene símbolos que representan las primitivas que constituyen al patrón. El conjunto de símbolos del lenguaje se conoce como su **alfabeto:**

$$\text{alfabeto} = \{a_0, a_1, \ldots, a_n, \ldots\} \Leftrightarrow \text{conjunto de primitivas}$$

De esta manera, el conjunto de primitivas que describen al patrón se asocia en forma única con el alfabeto del lenguaje de descripción de patrones. En este contexto, un patrón es una **frase** del lenguaje, es decir, una concatenación de símbolos de su alfabeto, siendo el orden de

aparición de los mismos la información estructural del patrón:

sea A el alfabeto de un lenguaje de descripción de patrones

$$\text{patrón} \Leftrightarrow \text{frase} = a_i a_j a_k \cdots, \quad a_n \in A$$

Un conjunto de primitivas puede describir un sinnúmero de patrones, dependiendo de la forma en que éstas sean agrupadas y ordenadas. Todos estos patrones forman un conjunto infinito denotado como A^*; cada subconjunto (frase) posible de símbolos del alfabeto, sin restricción en el número de repeticiones de un mismo símbolo en una frase dada es miembro de A^*. Naturalmente la frase $\varnothing$, el conjunto vacío, es también un miembro de A^*. Para denotar la exclusión del conjunto vacío se utiliza $A^+ = A^* - \varnothing$.

Claramente, la especificación de un conjunto A^* a partir de un alfabeto de primitivas no es adecuada para resolver el problema de clasificación de patrones, pues el lenguaje que efectivamente represente a una clase de patrones dada será, a lo más, un subconjunto de A^* . Para acotar este subconjunto de frases e introducir la información estructural de la clase de patrones se recurre a la especificación de una **gramática generativa**, la cual define completamente a un lenguaje. Una gramática generativa es un conjunto G definido como:

$$G = \{N, T, P, S\}$$

donde:

N es un alfabeto de **símbolos no terminales**

T es un alfabeto de **símbolos terminales**

con $N \cap T = \varnothing$, $N \cup T = V$

$P \subset V^* \otimes V^*$ es el conjunto de **reglas de producción** de la gramática

$S \in N$ es el **símbolo inicial**

El alfabeto terminal T se utiliza para construir las frases del lenguaje $L \subset T^*$ descrito por la gramática generativa, de modo que los símbolos de T corresponden a las primitivas de los patrones. Los símbolos del alfabeto no terminal N son auxiliares para la formulación de las reglas de producción que contienen la información estructural del lenguaje. Una regla de producción $r \in P$ está definida como $r: \alpha \rightarrow \beta$, $\alpha \in V^+, \beta \in V^*$, indicando la sustitución de una ocurrencia de la frase o subfrase α por una ocurrencia de β.

Cuando una frase puede obtenerse a partir de otra mediante la aplicación de una o varias reglas de producción de una gramática generativa, se dice que aquella es **derivable** de ésta bajo la gramática dada. El lenguaje

generado por G es el conjunto de todas las frases derivables a partir del símbolo inicial $S \in G$, que son elementos de A^*:

$$L(G) = \left\{ x / x \in T^*, S \overset{*}{\underset{G}{\Rightarrow}} x \right\}$$

donde $\overset{*}{\underset{G}{\Rightarrow}}$ denota una derivación bajo G de x a partir de S, mediante la aplicación de una o más reglas de producción.

Las reglas de producción de la gramática generativa permiten establecer el criterio de pertenencia a una clase de patrones en la siguiente forma: " si un patrón problema pertenece al conjunto $L(G)$, entonces el patrón pertenece a la clase descrita por el lenguaje L asociado con la gramática generativa G ". Esta definición acota efectivamente al conjunto T^* de forma que resulte de utilidad para la clasificación, pues indica que si el patrón analizado, descrito con primitivas del lenguaje, cumple con una estructura o sintaxis claramente especificada, entonces el patrón forma parte de una clase dada.

Con el marco de referencia anterior puede replantearse la solución al problema del reconocimiento sintáctico de patrones (comparar estructuras) como la construcción de un método para probar la pertenencia de un patrón problema a un

lenguaje $L(G)$. Claramente esto requiere de la definición previa de primitivas y de una gramática generativa. El método de prueba es más bien un aspecto práctico del análisis sintáctico que se discutirá más adelante.

En el caso del ECG se han reportado diversos métodos de selección de primitivas. En un trabajo pionero, Horowitz [19] y Pavlidis y Horowitz [36] presentan una estrategia general de preprocesamiento basada en la aproximación de funciones a través de segmentos lineales y describen la conveniencia de su uso para señales discontinuas como el ECG en comparación con otras técnicas de aproximación como las series de potencias. Se resalta en este artículo el carácter indispensable del preprocesamiento de la señal como un paso de reducción del detalle (por ejemplo, el ruido de la línea de base) que permite enfocar la codificación en primitivas a la estructura global de la señal, facilitando su reconocimiento. La definición final de las primitivas se basa sin embargo en criterios experimentales.

En el ámbito de la inteligenica artificial, Birman [5] utiliza las salidas de un algoritmo clásico de compresión de datos (AZTEC [8]) para definir seis primitivas asociadas con las duraciones de los segmentos y las pendientes generadas en el código de compresión. Esta estrategia es fundamentalmente una aproximación por segmentos lineales también. En el artículo se destaca que los resultados del

análisis sintáctico aunados a una metodología de evaluación de reglas como apoyo adicional se mejoran considerablemente si en su formulación sólo se involucra información de complejos que se sabe estarán presentes en la señal por analizar. Por otro lado, Yáñez y cols. [43] han propuesto un esquema heurístico de clasificación de complejos normales que se basa también en la asociación de significado a cada tipo de segmento codificado de AZTEC.

Lee y cols. [27] reafirman el requerimiento de reducción del detalle como paso previo a la clasificación y proponen un método de caracterización de las ondas componentes del ECG basado en el análisis de la curvatura local de la señal. Aunque su estrategia de clasificación no es sintáctica, la metodología de descripción de las ondas puede adaptarse fácilmente a un esquema de extracción de primitivas. El análisis de curvatura es también utilizado por Trahanias y Skordalakis [41] para definir primitivas parametrizadas que representan picos de diferentes formas. Los parámetros de cada pico son utilizados como apoyo adicional para la clasificación sintáctica.

En la misma línea de la aproximación por segmentos, Udupa y Murthy [42] construyen sus primitivas en términos de la pendiente de los segmentos de recta con los que el algoritmo SAPA [21] codifica la señal de ECG. Los umbrales que definen la asignación de un segmento a una de siete

primitivas diferentes se determinan en forma experimental. Pietka [37] aborda el problema de definición de primitivas en forma similar.

Con una mezcla de técnicas clásicas de detección de onda R (diferenciación y prueba de umbral), Belfonte y cols. [4] localizan el complejo QRS y determinan medidas morfológicas del mismo, tales como la duración del pico y su amplitud, que sirven de apoyos adicionales para el esquema de reconocimiento sintáctico. Con estas medidas construyen un espacio bidimensional donde se ubica a cada complejo. Este espacio, dividido en 16 regiones, sirve para asociarle a cada complejo una de 16 primitivas que componen el alfabeto de su lenguaje.

La diversidad de acercamientos propuestos para la definición de primitivas indica que no es sencillo definir la solución única y mejor para este problema [42].

Una vez seleccionado el conjunto de primitivas que se utilizarán para describir las clases de patrones es necesario definir un lenguaje para cada clase. Los lenguajes pueden clasificarse según la jerarquía de Chomsky [1,35], que los ordena, de acuerdo a la categoría de restricciones impuestas sobre las reglas de producción de sus gramáticas generativas, en 4 tipos diferentes:

Tipo 0 (no restringida):

$$r:\alpha \rightarrow \beta, \quad \alpha \in V^{+}, \beta \in V^{*}$$

Tipo 1 (sensible al contexto):

$$r:xBy \rightarrow xzy, \quad x,y \in V^{*}, B \in N, z \in V^{+}$$

Tipo 2 (libre del contexto):

$$r:B \rightarrow z, \quad B \in N, z \in V^{+}$$

Tipo 3 (regular):

$$r:B \rightarrow cD \text{ ó } r:B \rightarrow c, \quad B,D \in N, c \in T$$

Se observa claramente que los lenguajes Tipo i $(i = 0,1,2,3)$ son menos restrictivos que los Tipo $i+1$, de manera que permiten hacer descripciones más ricas de la estructura de un patrón. En principio, se debería recurrir a estos lenguajes para atacar el problema del reconocimiento de patrones, sin embargo, los lenguajes de tipo 0 son **no decidibles**, pues no puede diseñarse un algoritmo que en un número finito de pasos resuelva el problema de pertenencia. Además, conforme se desciende en la jerarquía de lenguajes, el algoritmo de prueba se hace más simple, de modo que al atacar en forma práctica un problema de reconocimiento sintáctico debe establecerse un compromiso entre la simplicidad de implantación de algoritmos y la efectividad de clasificación de los mismos.

En la práctica, los investigadores involucrados con el análisis sintáctico del ECG se han inclinado por la utilización de lenguajes libres de contexto, sin quedar claras en sus escritos las razones de la elección. El común denominador en esos trabajos es que la plataforma de cómputo utilizada es suficientemente potente como para enfrentar el problema de pertenencia a estos lenguajes.

Trabajos destacados en este rubro son los de Trahanias y Skordalakis [41] y Udupa y Murthy [42]. En ambos casos se parte del conocimiento a priori de la señal y se formula íntegramente una gramática de complejos normales. Aunque con buenos resultados, éstos trabajos definen sendos parámetros que deben ser determinados experimentalmente y, aunque se discute abundantemente sobre las implicaciones de sus valores, resulta clara la vulnerabilidad de sus algoritmos en cuanto a su repetibilidad y adaptación a clases de patrones no conocidos con anterioridad. Papakonstantinou y Gritzali [33] y Skordalakis [38] realizaron extensiones de estos trabajos en aplicaciones de filtrado sintáctico.

En el sentido de proporcionar a los esquemas de reconocimiento cierto grado de adaptabilidad, Pietka [37] discute una técnica de aplicación en paralelo de pruebas de aceptación para varios lenguajes y afirma que las gramáticas que utiliza se construyen a partir de muestras reales del ECG analizado. Desafortunadamente esta afirmación queda

obscurecida por una deficiente descripción de la metodología utilizada.

Otro intento para integrar adaptabilidad al sistema de reconocimiento ha sido presentado por Albus [2], mediante la utilización de gramáticas estocásticas [6,15] . Este trabajo se orienta principalmente al modelado del sistema de conducción cardiaco y sólo su potencialidad como instrumento de clasificación es mencionada.

El trabajo de Belfonte y cols. [4] es el único, hasta donde la revisión realizada indica, que utiliza lenguajes Tipo 3 para su sistema. Las gramáticas utilizadas se definen con base en información experimental previa y se clama su capacidad de adaptarse ante nuevas entradas. Como en otros casos, la sustentación del sistema en varios parámetros experimentales indica su potencialidad de falla. Sin embargo, las posibles limitaciones del clasificador sintáctico se aligeran con el uso de análisis multi-derivación (aprovechando la redundancia de información en los canales de ECG) y la introducción de medidas cuantitativas de apoyo adicional.

CAPITULO 2

DESCRIPCION DEL PROBLEMA

Se ha discutido el proceso de clasificación de patrones basado en la caracterización morfológica a través del análisis estructural sintáctico. Básicamente, la técnica consiste en elegir un conjunto de primitivas, representar los patrones con ellas, definir gramáticas generativas que describan clases de patrones y construir un método que resuelva el problema de pertenencia. En el presente caso, la clase de patrones la representan los complejos QRS normales en el registro electrocardiográfico de un paciente sometido a un estudio ambulatorio de 24 horas.

La elección de las primitivas con las que se representará el ECG debe hacerse con una orientación tal que la codificación de los trazos se acople de la mejor manera posible al sistema de manejo de datos, manteniendo además la importante característica de reducir la cantidad de información de la señal. Esto resulta especialmente importante en los sistemas de electrocardiografía ambulatoria digital donde las condiciones de adquisición originan volúmenes de datos del orden de varios megabytes [44]. Naturalmente, cualquier sistema de éstos debe

incorporar de por sí algún esquema de compresión de datos [40].

Son varias las técnicas de compresión del ECG que se asemejan a las alternativas para la extracción de primitivas y la codificación de patrones descritas en el capítulo anterior. Se nota una marcada predilección por las aproximaciones lineales a trozos, debido a la forma típica de un ciclo de ECG. Sin embargo, la perspectiva de comprimir y codificar simultáneamente no se presenta en trabajos previos, pues en el interés de desarrollar técnicas generales de reconocimiento del ECG no se pone atención a la metodología particular de diagnóstico cardiológico en la que se utilizarán. Para el caso de la electrocardiografía ambulatoria digital, comprimir y codificar es sin duda la estrategia de elección, pues sobre estos sistemas está impuesto un requerimiento de mínimo consumo de potencia, que se traduce en poca capacidad de memoria, así como el requisito de capacidad de procesamiento en tiempo real [40]. Siendo indispensable comprimir el ECG, entonces la codificación de la información comprimida puede hacerse en línea sin perder mayor tiempo.

Si el problema de representación se resuelve simultáneamente con la compresión del ECG, entonces la definición de las primitivas depende ampliamente del método de compresión elegido. Por sus características [44], el

algoritmo AZTEC resulta ser un buen candidato para implantarse en un sistema digital de capacidad mínima, como lo es la unidad portátil que albergará todo el sistema de clasificación motivo de éste trabajo [24]. La elección de primitivas puede ser en consecuencia tan simple como la siguiente: segmento con pendiente cero, segmento con pendiente positiva, segmento con pendiente negativa, con lo que el alfabeto del lenguaje de descripción resulta de 3 símbolos únicamente. Otras variantes de elección son también posibles, a través de la división de los segmentos por su duración o amplitud.

Resulta importante, dado el contexto del problema, meditar también sobre la conveniencia de utilizar la aproximación convencional al problema de análisis, consistente en especificar a priori el lenguaje que describe a un complejo normal, tal como se ha hecho en los trabajos descritos en el capítulo anterior. La situación real es que nada se sabe sobre el "complejo QRS normal" del paciente que será estudiado, ni de ningún otro paciente, más que por información estadística promedio. Si esta información promedio no considera como normal la existencia de, por ejemplo, un bloqueo de rama derecha, entonces un trazo electrocardiográfico de un paciente que ha vivido años con este bloqueo será considerado anormal por el sistema de reconocimiento establecido. Para considerar este caso, y

muchos otros variantes del "complejo QRS normal" se requeriría construir una gramática altamente compleja que probablemente generara lenguajes no decidibles, provocando la imposibilidad práctica de la clasificación.

Resulta más atractivo un esquema adaptable de análisis sintáctico, en el que el sistema de reconocimiento se construya en base a información real de la morfología del complejo normal de cada paciente. Esto significa que **la gramática generativa debe inferirse** a partir de los datos reales [13,14]. También significa que el método de prueba de pertenencia debe ser construido en base a esta misma información. El requerimiento de adaptabilidad del sistema restringe el tipo de lenguajes que pueden utilizarse para describir los patrones, pues la inferencia gramatical ha sido desarrollada con cierto éxito sólo para lenguajes regulares y en mucha menor medida para lenguajes libres de contexto [13,14].

Suponiendo que los procedimientos de inferencia para cada uno de estos tipos de lenguaje implicaran una complejidad algorítmica equivalente y resultados comparables, debe considerarse un criterio adicional para elegir alguno de los dos como el lenguaje de descripción. Este criterio es la simplicidad de implantación computacional, necesaria si el sistema electrónico de electrocardiografía ambulatoria para el que se diseña la

técnica de clasificación contiene un procesador de capacidad limitada. Siendo éste el caso del presente trabajo, la elección de un lenguaje regular resulta la más adecuada.

Las reflexiones anteriores permiten establecer con mayor claridad la estructura de una posible solución al problema de reconocimiento de complejos normales, pudiendo entonces plantearse la siguiente hipótesis y el objetivo para este trabajo:

HIPOTESIS. La información obtenida mediante el algoritmo de codificación AZTEC es en sí misma una representación estructural de la señal de ECG, que puede analizarse por técnicas sintácticas y ser utilizada para diferenciar complejos normales y anormales.

OBJETIVO. Construir un sistema que, a partir de muestras reales de la señal de ECG de un paciente, obtenidas con un sistema digital de electrocardiografía ambulatoria y codificadas con AZTEC, permita inferir una gramática regular que corresponda a un lenguaje de descripción de patrones de los complejos QRS normales del paciente y que además realice la prueba de pertenencia.

CAPITULO 3

EL PROBLEMA DE PERTENENCIA EN LOS LENGUAJES REGULARES

La solución completa del problema de pertenencia de un patrón a un lenguaje dado se obtiene al constatar que la aplicación de las reglas de producción de la gramática asociada permite una derivación completa de la frase que representa al patrón y conocer al mismo tiempo esta derivación en forma explícita [6,15]. Este es el enfoque utilizado por los compiladores de lenguajes de programación para construir el código objeto de un programa y las técnicas utilizadas para implantarlo se conocen con el nombre genérico de **parsing** [1].

Sin embargo, para el caso del reconocimiento sintáctico de patrones puede no resultar de interés práctico conocer con detalle la derivación, sino que es deseable solamente saber si la derivación existe o no. Puesto en términos lingüísticos, se desea saber únicamente si la frase es **aceptada** o no por el lenguaje de descripción de patrones.

La prueba de aceptación en un lenguaje regular se realiza algorítmicamente mediante la construcción de un

autómata de estados finitos (AEF) [6], definido por el conjunto:

$$A = \{I, Q, \delta, q_0, F\}$$

donde:

I es el conjunto de **entradas**

Q es el conjunto de **estados permitidos**

δ es el conjunto de **transiciones**

$q_o \in Q$ es el **estado inicial**

$F \subset Q$ es el conjunto de **estados finales**

El AEF está caracterizado en todo momento por el estado $q_i \in Q$ en el que se encuentra. Originalmente el autómata se encuentra en el estado inicial q_0 y cambia a otros estados en respuesta a las entradas $a_i \in I$, de acuerdo a las transiciones definidas en el conjunto δ. Una transición de este conjunto está definida como $\delta(q,a) = q'$, $q, q' \in Q, a \in I$ e indica que si el autómata se encuentra en el estado q y es presentado con la entrada a, entonces el autómata cambiará de estado para permanecer en el q' mientras no haya otra entrada que le provoque una nueva transición. Para una sucesión de entradas $x = a_1 a_2 a_3 \cdots a_n$ puede definirse también la transición:

$$\delta(q,x) = \delta(\cdots\delta(\delta(q,a_1),a_2),a_n)$$

que abrevia la escritura de las transiciones sucesivas que sufre el autómata al ir recibiendo secuencialmente como entradas los elementos de x.

El conjunto de estados finales es utilizado para resolver el problema de aceptación. Si para una frase $x \in I^*$ resulta que $\delta(q_0,x) \in F$, entonces se dice que el autómata "acepta" la frase. El conjunto de todas las frases aceptadas por un AEF es en sí mismo un lenguaje. Dos representaciones habituales para un AEF, la tabla de transición y el diagrama de transición se muestran a manera de ejemplo en la Figura 1.

La utilidad del concepto del autómata de estados finitos es principalmente práctica ya que puede implantarse en forma sencilla en una computadora. Pero esta característica sería inútil si no fuera posible construir un AEF tal que el lenguaje que acepte fuera equivalente al lenguaje regular de descripción de patrones con el que se desea hacer la clasificación; es decir se requiere que:

$$L(A) \equiv L(G)$$

$$L(A) = \{x / x \in I^* \quad , \quad \delta(q_0,x) \in F\}$$

$$L(G) = \{x / x \in T^* \quad , \quad S \overset{*}{\underset{G}{\Rightarrow}} x\}$$

Para una gramática regular siempre es posible construir un AEF que cumpla con la anterior relación de equivalencia si se vigilan las siguientes condiciones [6,15]:

- se define $I \equiv T$
- para cada no terminal $A \in N$ de la gramática se incluye un estado posible q_A en el autómata
- se incluyen dos estados adicionales en el autómata: q_F (final) y q_T (trampa)
- se define $F = \{q_F\}$
- para cada regla de producción $X \rightarrow aY$ en la gramática, se incluye una transición $\delta(q_X, a) = q_Y$
- para cada regla de producción $X \rightarrow a$ en la gramática, se incluye una transición $\delta(q_X, a) = q_F$
- todas las transiciones no definidas con las dos reglas anteriores son de la forma $\delta(q, a) = q_T$

Con la observación de las reglas anteriores es sencillo obtener un AEF a partir de una gramática generativa. Este autómata puede entonces ser utilizado para desarrollar la labor de reconocimiento de patrones. Es posible que el autómata resultante sea no determinístico (con transiciones posibles a más de un estado con un mismo símbolo de

entrada), en cuyo caso es necesario desarrollar un procedimiento que lo transforme en uno determinístico.

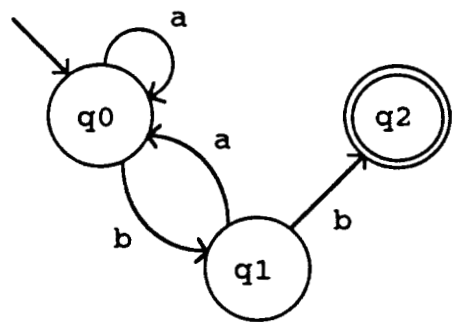
$$A = \{\{a,b\}, \{q_0,q_1,q_2\}, \Delta, q_0, \{q_2\}\}$$

$$\Delta: \quad \delta(q_0,a) = q_0 \; ; \; \delta(q_0,b) = q_1 \; , \; \delta(q_1,a) = q_0 \; , \; \delta(q_1,b) = q_2$$

(a)

TIPO DE ESTADO	ESTADO	a	b
inicial	q_0	q_0	q_1
transición	q_1	q_0	q_2
final o de aceptación	q_2	-	-

(b)



(c)

Figura 1. (a) Autómata de Estados Finitos. (b) Su Tabla de Transición. (c) Su Diagrama de Transición.

Para concluir es conveniente destacar que el problema de pertenencia, traducido a la aceptación por un autómata de estados finitos, se presenta como una alternativa altamente viable en una plataforma de cómputo como la del sistema para el que se desarrolló esta técnica de reconocimiento, pues el

algoritmo puede reducirse a representar la tabla de transiciones en la memoria del sistema y a recorrerla eficientemente con un modo de direccionamiento como el indexado.

CAPITULO 4

EL PROBLEMA DE LA INFERENCIA DE GRAMATICAS REGULARES

Resta por definir la forma en la que es posible construir una gramática generativa (o su autómata equivalente) a partir de las muestras reales del ECG. Formalmente el problema se expresa como la búsqueda de una gramática $G = \{N, T, P, S\}$ tal que dado un conjunto de frases $M \subset T^*$ se cumpla que $I \subset L(G)$.

De principio pueden ofrecerse dos soluciones triviales a este problema. Por ejemplo, para reconocer las frases del lenguaje $L = \{a, bc, abc\}$, se pueden construir los AEF que se muestran en la Figura 2.

Estas son dos soluciones que no resultan útiles para atacar el problema de reconocimiento, aún cuando son totalmente correctas. El autómata mínimo universal acepta al lenguaje L, pero también a cualquier otro lenguaje cuyo alfabeto sea idéntico al suyo. Por otro lado, el autómata máximo canónico es muy rígido ya que sólo acepta al lenguaje L y evita la generalización perseguida con la inferencia gramatical.

CAPITULO 4

EL PROBLEMA DE LA INFERENCIA DE GRAMATICAS REGULARES

Resta por definir la forma en la que es posible construir una gramática generativa (o su autómata equivalente) a partir de las muestras reales del ECG. Formalmente el problema se expresa como la búsqueda de una gramática $G = \{N, T, P, S\}$ tal que dado un conjunto de frases $M \subset T^*$ se cumpla que $I \subset L(G)$.

De principio pueden ofrecerse dos soluciones triviales a este problema. Por ejemplo, para reconocer las frases del lenguaje $L = \{a, bc, abc\}$, se pueden construir los AEF que se muestran en la Figura 2.

Estas son dos soluciones que no resultan útiles para atacar el problema de reconocimiento, aún cuando son totalmente correctas. El autómata mínimo universal acepta al lenguaje L, pero también a cualquier otro lenguaje cuyo alfabeto sea idéntico al suyo. Por otro lado, el autómata máximo canónico es muy rígido ya que sólo acepta al lenguaje L y evita la generalización perseguida con la inferencia gramatical.

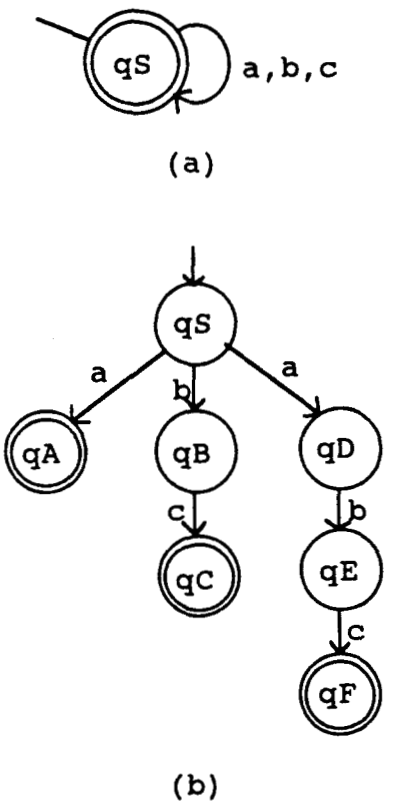


Figura 2. (a) autómata mínimo universal. (b) autómata máximo canónico. Ambos aceptan el lenguaje $L=\{a,ab,abc\}$.

Pero el número de soluciones al problema resulta en realidad mucho mayor, si consideramos la construcción de un **autómata derivado** a partir del autómata $A=\{I,Q,\delta,q_0,F\}$.

$A'=\{I,P,\delta',p_0,R\}$ es un autómata derivado de A si cumple con las siguientes condiciones:

- el conjunto de estados derivados P es una partición del conjunto de estados originales Q, definida como: $P=\{p_0,p_1,\ldots,p_r\},\ \ p_i \in Q^*, p_i \cap p_j = \varnothing \ \ \forall i \neq j$

- existe la transición $\delta(p_i,a)=p_j$, $p_i,p_j \in P$, si existen $q_k \in p_i$ y $q_l \in p_j$ tales que $\delta(q_k,a)=q_l$
- el conjunto p_0 contiene al estado inicial original q
- los estados finales están definidos como el conjunto siguiente: $R=\{p_i \in P / \exists q_k \in F, q_k \in p_i\}$

Claramente, el número de autómatas derivados de uno base es muy grande, dependiendo del número de particiones posibles que puedan hacerse de los estados originales. La afirmación de que la construcción de autómatas derivados incrementa el número de soluciones del problema se obtiene directamente de las condiciones anteriores, pues por definición se cumple que $L(A) \subset L(A')$, indicando con ello que un autómata derivado introduce un nivel de generalización a la gramática asociada con el autómata base. Por lo anterior, todos los autómatas derivados del autómata máximo canónico son soluciones al problema de reconocimiento.

Dado que el número de soluciones resulta muy grande y no existe una manera efectiva de evaluar que tan buena es una solución respecto a otra, es necesario recurrir a una decisión práctica que permita seleccionar una única partición del autómata canónico, para lo cual varios métodos prácticos, tanto supervisados como no supervisados, han sido descritos en la literatura [13,14].

Para el presente trabajo se ha elegido como algoritmo de inferencia una variante del método supervisado de Chomsky-Miller descrito en [6]. La selección del algoritmo se ha basado en su factibilidad computacional y en el hecho de que la generación del autómata de aceptación a partir del resultado del algoritmo es inmediata y directa. La heurística de la selección de la partición del autómata derivado consiste en la búsqueda automática de las posibles recursiones presentes en la muestra de patrones.

Se rastrean las muestras de "entrenamiento" en busca de subfrases repetidas, las que se suponen generadas por transiciones sucesivas a través de un mismo lazo en el autómata representativo. Cuando se ha elegido la subfrase que procede de la recursión más probable, según el número de repeticiones y ocurrencias, cada aparición de la subfrase es sustituida por un símbolo nuevo que represente la expresión regular $(subfrase)^+$ (que simboliza una o más repeticiones de *subfrase*), con lo que se completa un paso en la inferencia. El método se repite hasta que no es posible encontrar más recursiones.

En orden inverso al de sustitución de símbolos nuevos se reescriben las expresiones regulares obtenidas, de manera que éstas queden escritas únicamente en términos del alfabeto terminal de la gramática (las primitivas de los

patrones). El autómata de aceptación puede construirse de aquí en forma directa.

Aún cuando la anterior es una solución correcta y válida al problema de reconocimiento sintáctico de patrones persiste el riesgo de que la partición elegida con el método de inferencia no resulte del todo adecuada en la práctica. Este es un riesgo con el que se debe convivir si se opta por una aplicación de esta índole.

Sin embargo es posible todavía recurrir a la teoría de lenguajes formales para atenuar el problema. Una gramática generativa puede ser aumentada con **atributos** [6,32,33] para suplir las posibles deficiencias que la elección de primitivas y el autómata de aceptación tengan en la descripción cuantitativa de las longitudes de los segmentos o las combinaciones de éstos dentro del código de compresión de AZTEC. Esta ampliación con atributos es realmente un acercamiento al reconocimiento de patrones muy parecido a la construcción de un vector de características para la clasificación por distancia. Ya se ha comentado al inicio de este escrito que en ocasiones ambos puntos de vista pueden resultar complementarios. De hecho, los "apoyos adicionales" que se refirieron en el capítulo 1 al discutir el estado del arte en el reconocimiento sintáctico del ECG no son sino atributos para las gramáticas utilizadas por los autores. Una descripción formal de las gramáticas con atributos ha

sido propuesta en relación a la semántica de los lenguajes de programación [26]. Su extensión al caso del reconocimiento de patrones es directa.

CAPITULO 5

EL SISTEMA DE INFERENCIA GRAMATICAL

Ha quedado establecido que para resolver el problema de la detección de complejos QRS normales utilizando la perspectiva de la inferencia gramatical es necesario:

- disponer de un mecanismo de selección de las muestras de ECG a partir de las cuales se inferirá la gramática
- definir un esquema de traducción del código de compresión de AZTEC a las primitivas del lenguaje que se inferirá
- diseñar una versión computacional del algoritmo modificado de Chomsky-Miller que realice la inferencia a partir de la muestra de patrones introducida y que ofrezca como salida la expresión regular del lenguaje inferido
- construir, a partir de esta expresión regular, un autómata de estados finitos que acepte el lenguaje generado por la gramática inferida
- elegir y mantener un conjunto de atributos que refuercen el mecanismo de reconocimiento

Resulta también importante considerar el diseño de un ambiente de usuario adecuado para la selección de la muestra de patrones así como para la valoración del comportamiento del detector de complejos ante un registro largo de electrocardiografía ambulatoria. En este trabajo se han desarrollado diversos programas que ejecutan todas las tareas antes mencionadas.

Aún cuando el objetivo terminal fue construir un sistema de detección de complejos normales, el diseño del código se planteó de forma tal que los mecanismos de inferencia gramatical y de aceptación resultaran independientes de la aplicación, es decir, que no dependieran ni de la codificación de las primitivas ni del manejo particular de la aceptación. De este modo, futuras extensiones de la técnica sintáctica o bien su uso para otras aplicaciones resultarían problemas menos complejos. El código descrito en este capítulo se refiere únicamente al sistema general de inferencia gramatical desarrollado.

Alrededor de toda la programación del sistema de inferencia se elaboró un programa de aplicación interactivo (interfase gráfica guiada por eventos y basada en ventanas) que permite explorar las capacidades de la estrategia de detección. Es importante indicar que todo el código base de la interfase gráfica y del manejo de las ventanas fue

desarrollado en forma original. De este modo, un resultado colateral del trabajo fue la obtención de un conjunto de librerías que pueden usarse para construir interfases gráficas.

Como puede notarse, la idea constante en el diseño del código fue la posibilidad de reutilizarlo en otras aplicaciones. Para estos casos, las mejores herramientas de desarrollo son los lenguajes orientados a objetos. Por ello, toda la programación fue desarrollada en C++ (Borland C++ V3.1, Borland International 1992). Esta elección de lenguaje facilitó la escritura del código del sistema de inferencia, pues permitió utilizar librerías comerciales de manejo de estructuras de datos dinámicas (Tools.h++ Class Library V5.1, Rogue Wave Software 1992), las cuales abundan en la programación diseñada.

De esta forma, el sistema de inferencia gramatical se conceptualizó como una clase (llamada **GramaticaRegular**), que opera sobre frases de entrada representadas con cadenas de caracteres. Estas frases pueden ser suministradas de diversas maneras. El resultado de la acción de la clase sobre las frases es la construcción interna de una gramática regular y su autómata de aceptación asociado. La definición de la interfase pública de la clase hace que estos elementos sean accesibles para el usuario de la misma, siendo particularmente importante el acceso al autómata, de modo

que puedan implantarse procedimientos específicos de aceptación (prueba de atributos, por ejemplo). La Figura 3 muestra un esquema general de la clase GramaticaRegular, donde pueden observarse sus funciones miembro fundamentales, así como su relación con las entradas y salidas. El resto de funciones miembro son esencialmente auxiliares, para una descripción de los mismos puede recurrirse al listado del programa en el apéndice A.

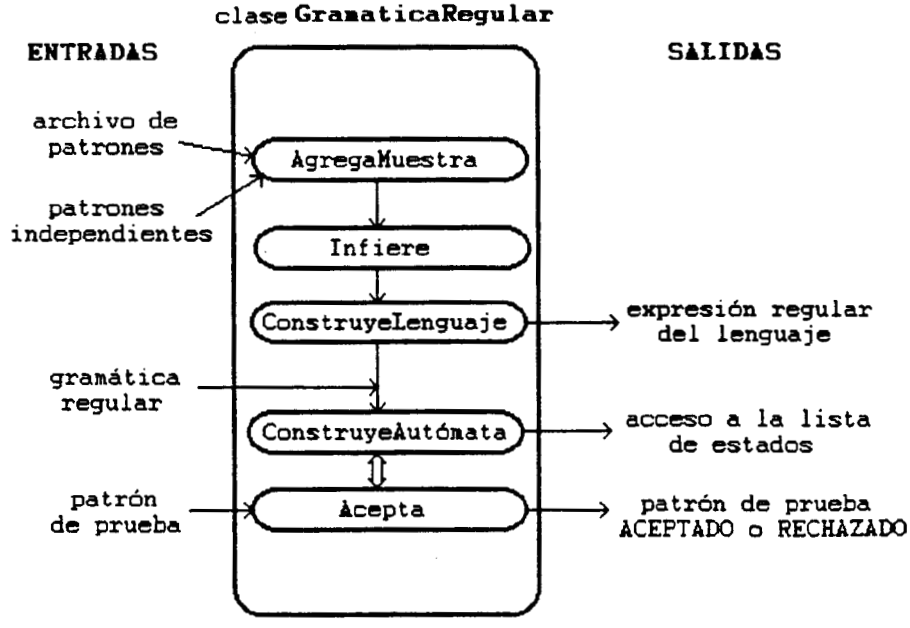


Figura 3. Elementos básicos de la clase GramaticaRegular

La clase GramaticaRegular contiene toda la funcionalidad del sistema de inferencia. Para utilizarla, el programa de aplicación debe declarar un objeto de esta clase, inicializándolo con alguno de los dos constructores ofrecidos para este fin.

En una aplicación de reconocimiento de patrones donde se desee inferir la gramática a partir de muestras seleccionadas una a una, el constructor que debe usarse es el constructor por omisión ***GramaticaRegular(void)***, que únicamente asigna espacios de memoria a las variables miembro que lo requieren. En este caso, es responsabilidad del programador indicar el momento en que la muestra se ha completado para proceder a la inferencia (llamando explícitamente a la función miembro ***Infiere(void)***). El segundo constructor, ***GramaticaRegular(archivo,ARCHIVO)*** obtiene un conjunto de muestras de un archivo en disco y realiza automáticamente la inferencia del lenguaje regular. Esta inicialización se ofrece para aquellos casos en los que la muestra de patrones ha sido elegida previamente.

Se ha incluido una opción de llamada al segundo constructor que tiene la forma ***GramaticaRegular(lenguaje,LENGUAJE)***, en la que se ignora todo el mecanismo de inferencia y se procede a construir el autómata a partir de la expresión regular definida en el parámetro *lenguaje*. Con esta opción, las aplicaciones que no

requieran de un proceso de inferencia pueden aún utilizar la maquinaria de la clase (por ejemplo, todas aquellas que se basen en una definición a priori de la gramática por utilizar).

La muestra de patrones que se utiliza como base de la inferencia es almacenada en una lista ligada simple, de manera que no hay restricción en el número de patrones que pueden formarla. Una vez elegida la muestra completa, es posible iniciar el procedimiento de inferencia. La descripción del algoritmo puede revisarse en la Figura 4.

Para mantener las tablas de subcadenas y la pila de traducciones a las que se hace referencia en dicha descripción se diseñaron las clases **InfoSubcad** y **TablaTraduccion**. La clase InfoSubcad registra información relativa al número de repeticiones consecutivas de una subfrase, así como el número de sus ocurrencias en la muestra base. La tabla de subcadenas es mantenida en una tabla de hash, de manera que las búsquedas e inserciones son muy eficientes. El campo llave de dicha tabla corresponde a las subcadenas, mientras que el campo de información se asocia con objetos de la clase InfoSubcad que mantienen la información de la subcadena asociada.

La clase TablaTraduccion se utiliza para relacionar subcadenas y símbolos nuevos durante la inferencia. Como la traducción subcadena-a-símbolo debe hacerse en orden inverso

a la sustitución final símbolo-a-subcadena resultó natural el manejo de la tabla de traducciones como una pila. De este modo fue necesario diseñar la clase a modo que permitiera que sus instancias pudieran ser introducidas a una pila.

El resultado de la aplicación exhaustiva de este algoritmo (en el ciclo REPITE...HASTA) es que cada frase de la muestra original es sustituida por una expresión regular que describe una familia de frases del lenguaje inferido. El lenguaje regular se obtiene finalmente de la unión de todas las expresiones regulares resultantes.

```
Infiere():

    REPITE
     PARA cada muestra base
          ·Busca subcadenas repetidas en la muestra
          PARA cada subcadena repetida
               ·Agrega información de la subcadena en la
                tabla de subcadenas
     SI la tabla de subcadenas tiene entradas
          ENTONCES  ·Busca la subcadena con más repeticiones
                     y mayor número de ocurrencias
                    ·Define la subcadena con nuevo símbolo
                    ·Mete definición a pila de traducciones
                    ·Cambia cada ocurrencia de la subcadena
                     por el nuevo símbolo
                    ·Limpia la tabla de subcadenas
                    ·Activa bandera de continuar
          OTRO ·Termina la inferencia
               ·Desactiva bandera de continuar
    HASTA bandera de continuar desactivada
    MIENTRAS   haya pila de traducciones
               ·Desapila definición
               ·Cambia cada ocurrencia del símbolo
                  por su frase
```

Figura 4. Seudocódigo de la función miembro Infiere, la cual obtiene un lenguaje regular a partir de una colección de frases muestra

Una vez inferido el lenguaje debe procederse a la elaboración del autómata de aceptación. Dado que el AEF es el actor principal en las aplicaciones de reconocimiento de patrones, se pensó en el diseño de una estructura de datos que lo manejara eficientemente. Es imposible saber a priori el número de estados que compondrán al autómata, pues cuando se ha ejecutado el proceso de inferencia se ha generado un autómata derivado del que no se conoce si sus transiciones son determinísticas o no determinísticas. Por esta razón, el AEF debe construirse en forma absolutamente dinámica.

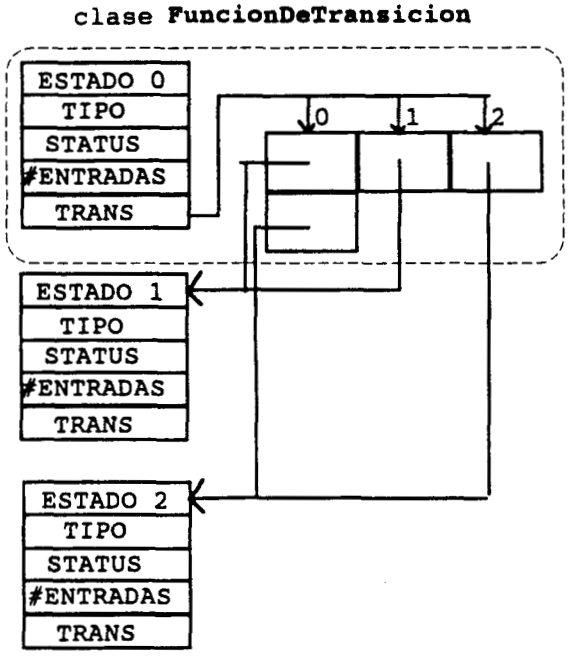


Figura 5. Estructura del AEF.

del AEF se eligió una estructura
; que asocien objetos de la clase
ı clase es utilizada para almacenar
iones asociadas con un estado, así
cación de su status. La Figura 5
la estructura de datos del AEF. Nótese
representación de un autómata en una
.

o que debe agregarse al autómata durante
asocia con una nueva instancia de la
nsicion. Una descripción en seudocódigo
creación del autómata se muestra en la
:ación del estado inicial se hace con una
al constructor de la clase. Luego, para cada
ar que conforma al lenguaje se construye una
ısertando estados según se requiera; para
ıentemente las conexiones de los estados, ya
.eran retroalimentación o no, se ha escrito un
o recursivo ***HazAEF(...)*** que es con el que en
ıino se construye el autómata. Este algoritmo
o a uno los símbolos de la expresión regular que
rocesando de modo que cuando encuentra un '(' se
ına autollamada para reiniciar el análisis con la
esión incluida en el paréntesis. La ocurrencia de
voca la conexión retroalimentada del autómata al

icial asociado con la sub-expresión que termina.
: otro símbolo provoca la creación de un nuevo
su inserción en forma serial en la cadena actual de
ión.

n el autómata construido, los siguientes pasos son
ente de adecuación. El más importante es el de resolver
posibles ambigüedades consistentes en haber generado
siciones no deterministicas. La transformación de un AEF
determinístico a uno determinístico se realiza mediante
construcción de un nuevo autómata, que incluye todos los
stados del AEF original no determinístico pero en el que
por cada conjunto de estados que forman una transición
ambigua se inserta un estado nuevo. Las transiciones de cada
nuevo estado se definen como la unión de las transiciones de
los estados originales que lo conforman. El paso de definir
las transiciones de los estados nuevos puede generar nuevas
transiciones ambiguas, en cuyo caso el proceso de
redefinición se repite iterativamente hasta que se han
resuelto todas las transiciones conflictivas. El algoritmo
de redefinición de estados ha sido implantado en la función
miembro ***ResuelveNoDets***(*void*). El último paso de adecuación
es la reducción del autómata por eliminación de los estados
a los que el AEF determinístico obtenido no podrá llegar
nunca. Esta eliminación se realiza con un recorrido

recursivo de toda la tabla de transiciones en la función miembro ***MarcaEstadosAccesibles****(void)*.

```
(a) ConstruyeAutomata():

·Crea el Estado Inicial
PARA cada expresión regular en el lenguaje
     ·HazAEF(exp regular,estado inicial,estado inicial)
·Resuelve transiciones no determinísticas
·Crea el Estado de Trampa
·Completa el autómata con transiciones al estado trampa
·Elimina estados inaccesibles

(b) HazAEF(exp,est_ini,est_fin):

PARA cada símbolo en exp
     ·Elimina de exp el símbolo analizado
     SI el símbolo es '('
        ENTONCES SI es el primer símbolo
                    ENTONCES HazAEF(exp,est_ini,est_fin)
                    OTRO HazAEF(exp,est_ini,est_fin+1)
     SI el símbolo es ')'
        ENTONCES Define una transición de est_fin a
                    est_ini para el símbolo actual
        OTRO crea un nuevo estado
             ·Define una transición de est_fin al nuevo
                estado para el símbolo actual
```

Figura 6. Seudocódigo de las funciones miembro (a)ConstruyeAutómata y (b)HazAEF, que definen un AEF a partir de la expresión regular de un lenguaje

La información de las transiciones de los estados es mantenida en los objetos de la clase FuncionDeTransicion dentro de un vector ordenado de listas ligadas simples. Cada lista ligada corresponde a la(s) transición(es) que un estado presenta ante un símbolo de entrada. El uso de una lista ligada simple se justifica debido a que es posible que

el autómata resulte no determinístico y que deban por ello incluirse más de una transición para una misma entrada.

La existencia del vector ordenado de listas permite un recorrido eficiente del autómata, pues cada posición dentro del vector puede asociarse a un único símbolo terminal. Si los símbolos terminales se ordenan a su vez en una cadena, entonces la relación de posiciones en la cadena y el vector debe ser respetada. De este modo, cuando se desea saber a que estado transitará el autómata dado un símbolo de entrada el procedimiento de solución es simplemente ubicar el índice dentro de la cadena de terminales que corresponde a la entrada (esta cadena es una variable miembro de GramaticaRegular) y leer la lista de transiciones que se encuentra en la posición índice del vector del estado actual.

Una vez construidos el lenguaje y la gramática es posible usar la capacidad propia de la clase GramaticaRegular para realizar pruebas de aceptación. Para ello, se ha incluido en la definición de la clase una función miembro pública llamada ***Acepta****(const char * patrón)*. Esta función recorre el autómata hasta utilizar todos los símbolos del *patrón*, reportando al final si el estado en el que quedó el autómata es de aceptación o no. ***Acepta*** es un buen ejemplo de la facilidad con la que se recorre el AEF dada la estructura que se utilizó para construirlo, sin

embargo, es poco probable que sea utilizada para problemas de reconocimiento de patrones, que típicamente extienden las gramáticas con atributos numéricos. Siendo este el caso en el presente trabajo es necesario ahora describir la versión particular del algoritmo de aceptación desarrollado.

CAPITULO 6

CODIFICACIÓN DE PRIMITIVAS Y ALGORITMO DE ACEPTACIÓN

La herramienta de inferencia gramatical desarrollada en este trabajo es fundamental pero no suficiente para abordar el problema de la detección de complejos normales en registros largos y contínuos de ECG como los de la electrocardiografía ambulatoria. Por su naturaleza general, la clase GramaticaRegular debe ser complementada con código específico de la aplicación donde va a utilizarse.

La programación complementaria debe abarcar el problema de la codificación de las primitivas, así como el del método particular que se requiera para la prueba de aceptación. Esto se traduce, para el problema de detección de complejos, a la elaboración de un esquema de traducción del código de compresión de AZTEC y a la selección y manejo de atributos que refuercen el esquema de reconocimiento.

En referencia al código de compresión generado por AZTEC es útil recordar que solamente existen dos tipos de datos de salida, los segmentos de pendiente cero y los de pendiente diferente de cero. La traducción directa de esta información genera solamente dos primitivas:

- la primitiva SEGMENTO (S), que se asocia con todos los segmentos de pendiente cero
- la primitiva PENDIENTE (P), que se asocia con todos los segmentos de pendiente diferente de cero

Es posible, sin embargo, aumentar el número de símbolos de la gramática de manera que la información reducida que la traducción anterior ofrece se vea enriquecida. El problema real es cuál de diversas opciones de traducción resulta la más adecuada para la aplicación particular y la respuesta a esta pregunta requiere necesariamente de trabajo experimental previo.

Con lo anterior en mente se diseñó un mecanismo de traducción de los datos de AZTEC a símbolos terminales (caracteres) que pudiera ser modificado sin dificultad. Para construir un conjunto de primitivas, el programador debe suministrar unicamente una lista de definiciones del estilo **PRIMITIVA**(condición,caracter), escrita entre los símbolos **TRADUCTOR** y **FIN_TRADUCTOR**. Esta estructura se muestra en la Figura 7(a) para la traducción directa a dos primitivas discutida anteriormente.

La simplicidad de esta definición se basa en el uso extensivo del preprocesador del lenguaje C++. Los símbolos TRADUCTOR, PRIMITIVA y FIN_TRADUCTOR son "macros" que el preprocesador expande en tiempo de compilación, definidos

para generar la declaración de un procedimiento llamado *Traductor(unsigned i)* que interpreta la *i*-ésima muestra de ECG comprimido, aplicando las condiciones definidas en cada PRIMITIVA y regresa el caracter asociado con la condición que se evalúe verdadera. En caso de que las definiciones de primitivas no resulten suficientes para interpretar una muestra, el traductor regresa el caracter '?'.

```
TRADUCTOR
   PRIMITIVA(segmento de pendiente cero,'S')
   PRIMITIVA(segmento de pendiente no cero,'P')
FIN_TRADUCTOR
```

(a)

```
TRADUCTOR
   PRIMITIVA(SEGMENTO,'S')
   PRIMITIVA(PENDIENTE,'P')
FIN_TRADUCTOR
```

(b)

```
#define TRADUCTOR       unsigned Traductor(unsigned i) {
#define PRIMITIVA(c,s)  if(c) return(s);
#define FIN_TRADUCTOR   return('?'); }

#define SEGMENTO        (dato*[i].dur <= 127)
#define PENDIENTE       (dato[i].dur > 127)
#define ANT_INF         (dato[i-1].pos < dato[i].pos)
#define PEND_POS        PENDIENTE && ANT_INF

* dato[] es el arreglo de datos codificados de ECG.
```

(c)

Figura 7. (a)Esquema para la codificación de primitivas, desde el punto de vista del programador. (b)El mismo esquema, usando los macros definidos. (c)Algunas definiciones de los macros utilizados.

Es claro que las condiciones no pueden ser establecidas en forma coloquial como en el caso de la Figura 7(a), pues éstas deben ser sintácticamente válidas en lenguaje C++. Con la intención de simplificar la definición de condiciones, una gran cantidad de éstas se han definido en forma de macros. La Figura 7(b) muestra la versión del codificador de primitivas usando los macros disponibles, algunos de los cuales se muestran en la Figura 7(c).

Con este método, el cambio de primitivas es rápido y las modificaciones al código transparentes al programador de la aplicación, además de que la definición de nuevas condiciones es bastante simple y directa.

Para el método de prueba de aceptación, implantado en el procedimiento ***Reconoce(void)***, se han incluido pruebas de atributos de extensión para la gramática, los cuales deben ser evaluados al mismo tiempo que el estado en el que se encuentra el autómata luego de cada transición.

Los atributos utilizados, la duración del complejo QRS (DQRS) y su amplitud diferencial máxima (ADM), han sido elegidos en función de la experiencia reportada en los trabajos que se han comentado anteriormente y en base a la noción de que éstas medidas caracterizan fundamentalmente al complejo en relación con el resto de las ondas del ciclo de ECG, pero el manejo que de éstos atributos se hace en el

presente trabajo es notablemente diferente al de los usos reportados.

Aprovechando que todo el sistema de reconocimiento parte de la definición de una muestra de patrones, los valores normales de DQRS y ADM se definen como el promedio de los correspondientes valores en todos los segmentos de ECG elegidos como muestra base, eliminando con ello el requerimiento de la definición de parámetros y/o umbrales a priori. Las desviaciones estándar de los datos pueden ser utilizadas como tolerancias al momento de la decisión de aceptación. Una secuencia de segmentos identificada como estructuralmente correcta por el autómata de aceptación puede o no ser definida como un complejo QRS normal dependiendo de si sus medidas DQRS y ADM caen o no dentro de los rangos permitidos. El procedimiento de aceptación diseñado se describe en forma de seudocódigo en la Figura 8. Como se observa, el procedimiento de aceptación es básicamente el mismo que utiliza la función miembro *Acepta()*, excepto por tres rasgos distintivos que deben ser apreciados.

El primero es la consideración de que el registro comprimido de electrocardiografía ambulatoria corresponde, para la duración comparada de un único complejo QRS, a una frase de duración "infinita". Esto acarrea dos problemas: primero, si se considera todo el trazo como una sola frase,

el autómata nunca detectará nada. Segundo, la primera vez que el autómata caiga en un estado de aceptación o de trampa ya no saldrá de ahí y el resto de la señal sera ignorado como no perteneciente a la clase de patrones estudiada.

La solución a estos problemas se obtiene al incluir en el código un mecanismo de reestablecimiento del autómata (regresándolo al estado inicial) y de determinación de la "sección válida" del trazo, definida como la secuencia específica de primitivas que serán tomadas en cuenta para la evaluación de los atributos. Del seudocódigo de la Figura 8 puede apreciarse que el reestablecimiento del autómata se genera cuando llega a un estado de trampa o aceptación, así como al obtenerse una sección válida cuya duración exceda al atributo DQRS.

El segundo rasgo que distingue al procedimiento de aceptación es que la afirmación de que se ha detectado un complejo se condiciona a los valores de DQRS y ADM para la sección válida de trazo que se ha procesado hasta el momento en que el autómata llega a un estado de aceptación.

Por último, debe resaltarse que se ha incluido en el algoritmo de aceptación el conocimiento previo consistente en que en una línea de base normal (sin ruido) sólo se obtendrán, del algoritmo AZTEC, segmentos de pendiente cero. Para evitar secciones válidas que contengan exceso de línea de base (con lo que se correría el riesgo de siempre exceder

DQRS) se recurre a una bandera que se activa sólo cuando hay la posibilidad de que la sección válida contenga información útil. Mientras la bandera está inactiva (sucesión de segmentos de pendiente cero), la sección válida se redefine como el último segmento codificado.

```
Reconoce():

·Autómata en estado inicial
·Haz nula la sección válida
·Desactiva bandera
MIENTRAS haya datos comprimidos
    ·Traduce el dato actual vía Traductor()
    ·Actualiza la sección válida
    ·Calcula atributos
    ·Evalúa transición dado el símbolo obtenido
    SI el estado actual es de trampa
      ENTONCES autómata en estado inicial
                ·Haz nula la sección válida
                ·Desactiva bandera
    SI el estado actual es de aceptación
      ENTONCES SI se cumplen atributos DQRS y AMD
                    en la sección válida
                  ·Indica detección de QRS
                ·Autómata en estado inicial
                ·Haz nula la sección válida
                ·Desactiva bandera
    SI el estado actual es de transición
      ENTONCES SI el dato actual tiene pendiente no cero
                 ENTONCES activa bandera
                 OTRO SI la bandera está desactivada
                        ENTONCES haz la sección válida
                                   igual al símbolo
                                 ·Calcula atributos
                 SI se excede atributo DQRS
                   ENTONCES autómata en estado inicial
                            ·Haz nula la sección válida
                            ·Desactiva bandera
    SI el estado actual es el inicial
      ENTONCES desactiva bandera
```

Figura 8. Prueba de aceptación

CAPITULO 7

PRUEBAS

El sistema de reconocimiento de patrones que ha sido descrito se evaluó en relación a tres aspectos principales:

- su funcionalidad general como un algoritmo de inferencia de gramáticas regulares a partir de muestras de ECG provenientes de registros de electrocardiografía ambulatoria
- el grado de diferenciación que ofrece para las morfologías anormales
- su vulnerabilidad a cambios de la muestra de entrada

Aún cuando para la valoración de cada uno de los aspectos anteriores se efectuaron pruebas específicas sobre diferentes registros de ECG, se observó también una metodología general.

Cada registro analizado fue comprimido previamente mediante la aplicación del algoritmo AZTEC, utilizando un umbral de compresión igual a cinco. De cada registro comprimido se eligió en forma aleatoria una muestra de seis complejos QRS normales que sirvió como entrada al algoritmo

de inferencia gramatical. Cuando se requirió realizar comparaciones entre dos tratamientos efectuados sobre el mismo registro, la muestra de entrada fue la misma. En todos los casos, con el propósito de estandarizar la toma de muestras, la región identificada como complejo QRS abarcó desde el comienzo del segmento PQ hasta el final del segmento ST del correspondiente ciclo de ECG.

La definición de primitivas se realizó, dependiendo de la prueba particular, en una o más de las siguientes formas, usando el procedimiento ***Traductor()*** descrito anteriormente:

Primitivas I:
S = segmento de pendiente cero
P = segmento de pendiente mayor que cero
N = segmento de pendiente menor que cero

Primitivas II:
M = segmento de pendiente cero, precedido de una primitiva P y seguido por una N
S = segmento de pendiente cero
P = segmento de pendiente mayor que cero
N = segmento de pendiente menor que cero

Primitivas III:
M = segmento de pendiente cero, precedido de una primitiva P y seguido por una N
V = segmento de pendiente cero, precedido de una primitiva N y seguido por una P
S = segmento de pendiente cero
P = segmento de pendiente mayor que cero
N = segmento de pendiente menor que cero

Una vez conocidas las primitivas y la muestra de patrones se procedió a la obtención del lenguaje inferido y a la construcción del autómata de aceptación. Asímismo se calcularon los valores promedio y desviaciones estándar de

los atributos DQRS y AMD de los patrones en la muestra de entrada. Se definió como criterio de aceptación que los atributos de la sección válida quedaran incluidos en un intervalo de tres desviaciones estándar alrededor de la media de los respectivos atributos de las muestras.

Finalmente, se eligieron diversos parámetros como indicadores del proceso de reconocimiento aplicado a un registro de ECG dado. Dichos parámetros se definen en la Tabla 1.

PARAMETRO	DEFINICION
DET_TOT	número de eventos detectados por el autómata
FP	número de detecciones de complejo indicadas por el autómata que resultaron erróneas ("falsos positivos")
FN	número de omisiones del autómata en la detección de complejos ("falsos negativos")
$QRS = DET\_TOT - FP$	número de complejos detectados correctamente
$\%DET = 100 \times \frac{QRS}{QRS + FN}$	porcentaje de complejos detectados correctamente en relación al total de complejos del registro analizado
$DFP = \frac{FP}{DET\_TOT}$	fracción de detecciones indicadas por el autómata que son erróneas

Tabla 1. Parámetros de evaluación de los autómatas de aceptación.

Valoración de la funcionalidad general del sistema. Se procesaron los registros de electrocardiografía ambulatoria obtenidos de dos sujetos sin historia previa de arritmias,

mediante una unidad portátil desarrollada dentro del marco del proyecto que incluye a la presente tésis, y cuyas características han sido reportadas previamente [24]. La unidad portátil entrega la información codificada según el algoritmo AZTEC, con una resolución de 7 bits y frecuencia de muestreo de 200 Hz.

De cada registro (duración promedio 2 horas) se eligió en forma aleatoria una sección de aproximadamente cinco minutos de duración, como muestra representativa para el análisis. Los trazos de las secciones elegidas, denominados aquí Registro I y Registro II, que fueron usados como muestra de entrada se presentan en las figuras 9 y 10.

Cada sección fue procesada bajo doce condiciones distintas, generadas por las combinaciones de uno de tres conjuntos de primitivas con uno de cuatro métodos de evaluación de los atributos de la sección válida, a saber:

a) prueba de aceptación sin validación de atributos
b) prueba de aceptación + validación de ambos atributos
c) prueba de aceptación + validación de AMD únicamente
d) prueba de aceptación + validación de DQRS únicamente

Los resultados de esta prueba se utilizaron para tomar una decisión sobre cuál o cuáles eran las mejores configuraciones del sistema de reconocimiento.

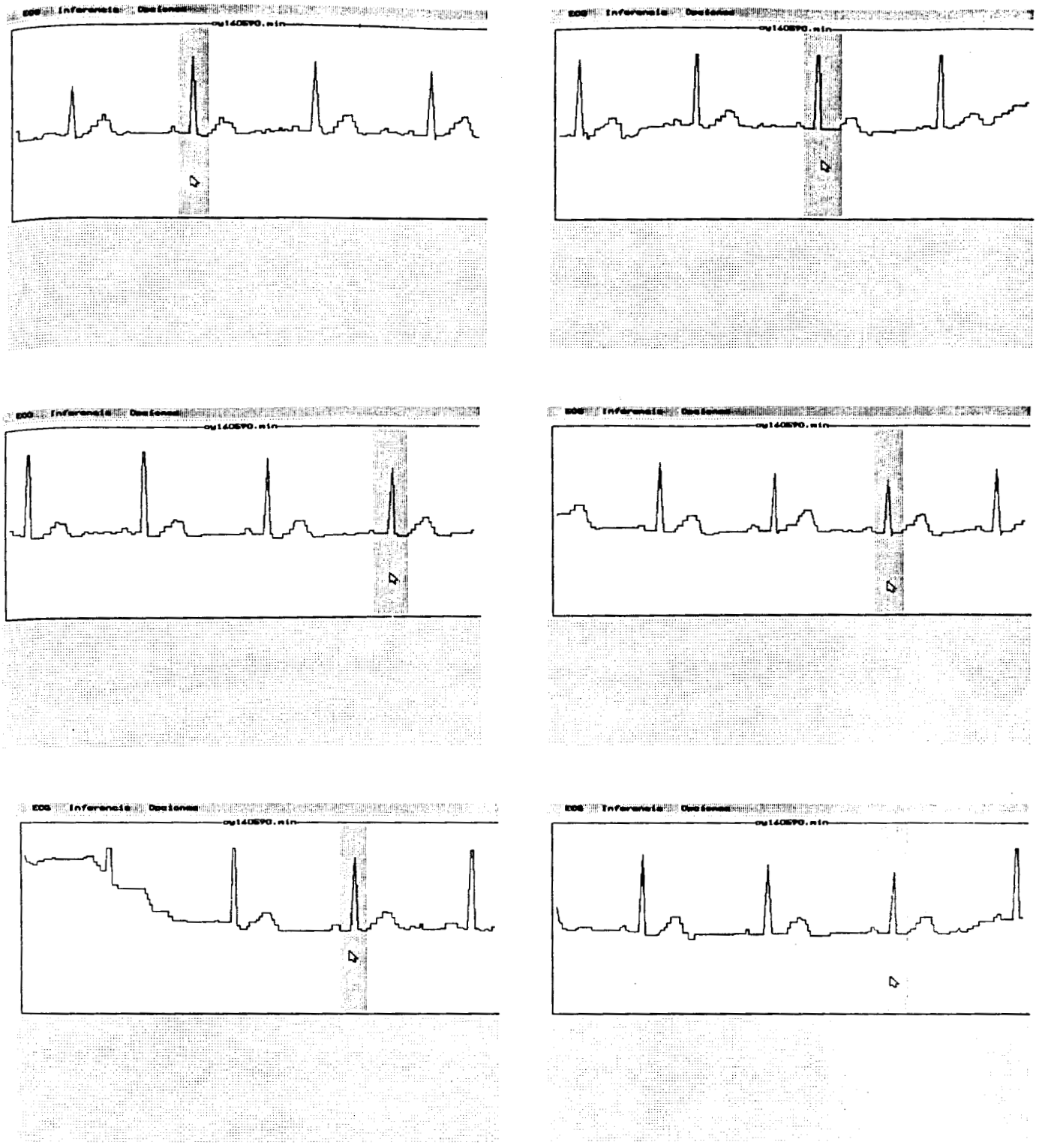


Figura 9. Patrones muestra seleccionados en el registro I (total: 187 complejos normales).

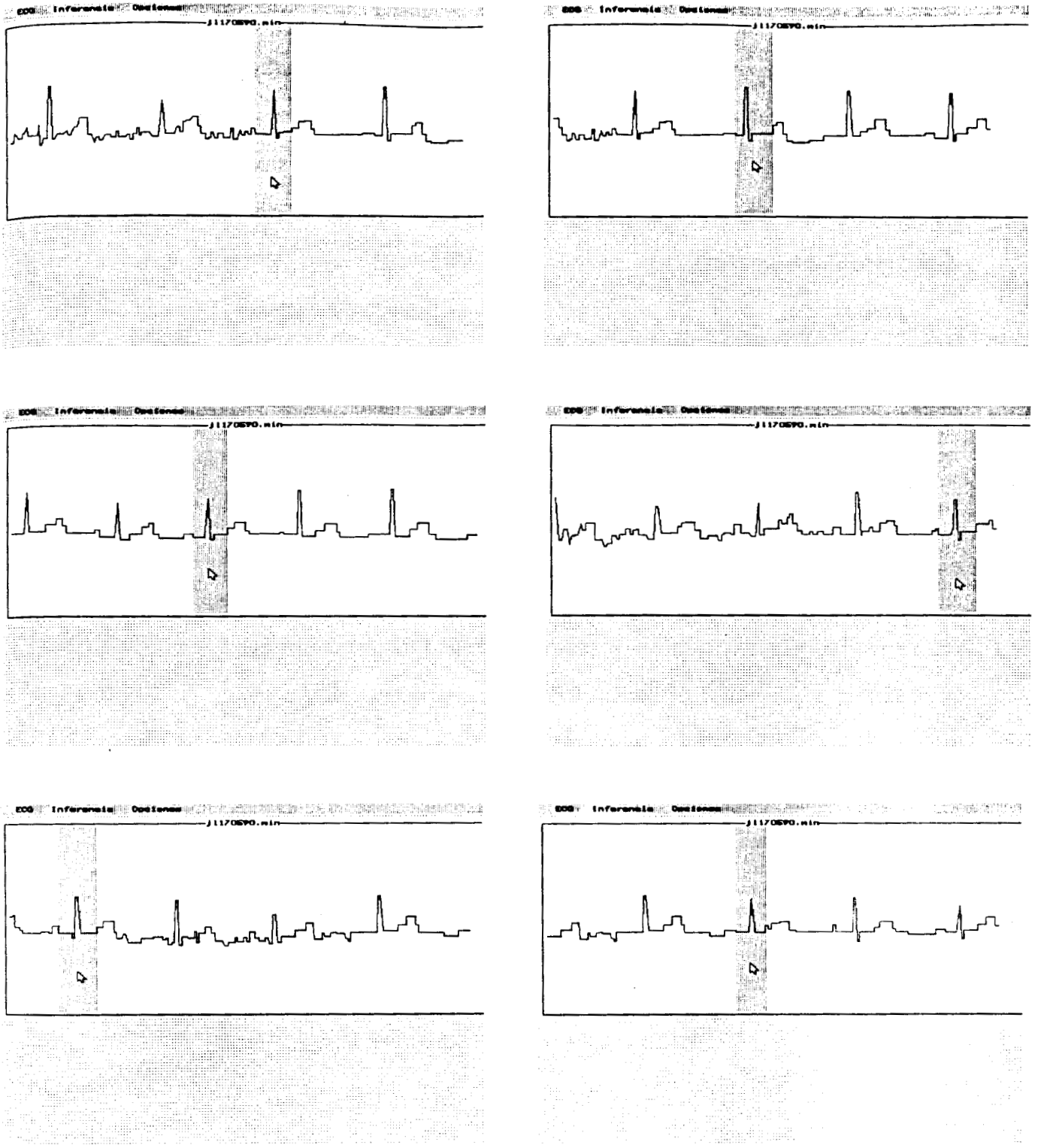


Figura 10. Patrones muestra seleccionados del registro II (total : 284 complejos normales).

Discriminación de Morfologías Anormales. Se procesaron cinco registros de ECG simulado (simulador digital BIOTEK) con duración de un minuto cada uno, adquiridos en una computadora personal a una frecuencia de muestreo de 200 Hz y resolución de 12 bits. El conteo promedio de complejos normales en cada registro fue de 38. El uso del simulador permitió la inclusión a voluntad de ciclos anormales en el registro de ECG. Las "arritmias" simuladas fueron dos tipos de contracción prematura ventricular, bradicardia, taquicardia y fibrilación ventricular. Asimismo se introdujeron, en algunos registros, tanto ruido de línea de base simulado, como artefactos por movimiento de electrodo. Algunos trozos de estas señales se muestran en la Figura 11. Cada registro se proceso con la configuración de sistema sugerida por la prueba anterior y se tomó particular atención a las detecciones FP en las zonas conflictivas.

Vulnerabilidad al Cambio de Muestras. Para cada registro real de los utilizados en la prueba inicial se construyeron cuatro nuevos autómatas con muestras diferentes tomadas igualmente al azar. Los parámetros generales estudiados fueron evaluados para cada sistema de aceptación.

CAPITULO 8

RESULTADOS Y DISCUSION

Las pruebas realizadas arrojaron resultados que, como se discute a continuación, apoyan la afirmación de que el sistema de reconocimiento de patrones propuesto es una buena alternativa de solución al problema de detección de complejos QRS normales, en la electrocardiografía ambulatoria digital.

La Figura 12 muestra algunos de los lenguajes inferidos con los registros de ECG ambulatorio. Se observa que en general las expresiones de los lenguajes son simples, aunque en algunos casos hay ocurrencias del símbolo '?' que refleja errores en el algoritmo de codificación utilizado (por ejemplo, segmentos de pendiente cero que se marcan como de pendiente positiva).

Las tablas 2 y 3 condensan información relativa a la prueba de funcionalidad del sistema de reconocimiento de patrones. El porcentaje de detección medio para todas las pruebas realizadas fue de 72%, aunque el rango máximo alcanzó el 94.7%. Se han reportado en la literatura diversos algoritmos no sintácticos que superan estas cifras de detección [12]. Podría considerarse por ello que el presente

sistema tiene un desempeño pobre. Sin embargo, se deben considerar algunos factores adicionales como los que se mencionan a continuación.

Registro I	$S^+PNS^+ + S^+PS^+NS^+ + S^+PNPS^+$
Registro II	$S^+?S^+\left(NS^+P^+\right)^+S^+ + S^+P^+NS^+ + S^+P^+S^+NS^+$

PRIMITIVAS:
M segmento de pendiente cero, precedido de una primitiva P y seguido por una primitiva N
V segmento de pendiente cero, precedido de una primitiva N y seguido por una primitiva P
S segmento de pendiente cero,
P segmento de pendiente mayor que cero
N segmento de pendiente menor que cero

Figura 12. Algunos lenguajes inferidos para las muestras de ECG ambulatorio.

Desde el punto de vista de la electrocardiografía ambulatoria los complejos normales son información redundante que se desea descartar. Esto es aún más cierto en el caso de la electrocardiografía ambulatoria digital, donde la economía en el uso de memoria es crucial. No detectar un complejo normal significa interpretarlo como uno anormal, gastando espacio de almacenamiento. Esto quiere decir que las deficiencias en la detección de complejos QRS (los falsos negativos) pueden subsanarse con una reserva de memoria de suficiente capacidad como para contender con el problema. Sin ser la mejor solución, es una salida posible.

No sucede lo mismo con los falsos positivos, pues si el sistema está diseñado para descartar la información reconocida como normal se corre el riesgo de que, ante una detección falsa, se ignore información que pudiera ser importante. Desde esta perspectiva, el problema que debe resolverse es la minimización de los falsos positivos. En este terreno, el sistema propuesto tiene un comportamiento destacado, tal y como puede deducirse al observar las fracciones de detecciones erróneas reportadas. Vale la pena indicar que no es común encontrar en los artículos sobre el tema una medida de las detecciones falsas positivas de los algoritmos propuestos.

Sin embargo es importante reconocer, según se ve comparativamente entre los diversos métodos usados para probar la aceptación, que el simple análisis estructural no resulta suficiente para abatir el número de falsos positivos. La inclusión de atributos en el análisis sintáctico queda así más que justificada. En particular se observa en las tablas 2 y 3 que el atributo responsable de la eliminación de los falsos positivos es la amplitud máxima diferencial (AMD). Dado que los falsos positivos son normalmente causados por ruido, es claro que el atributo AMD es suficiente para poner un umbral de significancia a trozos de señal que en forma azarosa tengan una composición

estructural similar a la de un complejo normal, pero no así su tamaño.

También se observa en estas tablas que, aparentemente, el atributo de duración del complejo (DQRS) es un atributo prescindible. Sin embargo, reflexionando sobre la naturaleza de las señales con las que se hicieron las pruebas (registros procedentes de sujetos asintomáticos), podemos concluir que la forma promedio del ECG no varió, de modo que el parámetro duración se hizo irrelevante. El atributo DQRS puede probarse de utilidad en casos reales de arritmia.

Otra condición que es notable es la existencia de un cierto orden en la capacidad de detección de una gramática, que es dependiente del conjunto de primitivas elegido. Si se observa con atención, para todos los métodos de prueba de aceptación, la codificación con las primitivas M,S,P y N produjo los mejores resultados globales. Las otras codificaciones, SPN y MVSPN, difieren de la primera en el grado de generalización o especificidad que introducen a la gramática, respectivamente. Esto podría significar que el grado de detalle con el que se elija representar un problema de análisis estructural sintáctico debe de ser ponderado con cautela.

La Tabla 4 muestra el comportamiento del sistema de análisis sintáctico ante registros que contienen diversas arritmias y artefactos. En este caso, con trazos simulados y

libres de ruido los porcentajes de detección se incrementaron notablemente y se anularon las detecciones falsas positivas. Esto es un indicador de la sensibilidad al ruido que tiene el método sintáctico, lo cual sugiere la necesidad del preprocesamiento de la señal. Se debe destacar también la imposibilidad que el método tiene para evadir las espigas del artefacto de movimiento (electrodo desconectado), pues ante ellas, el autómata indica una aceptación. Un posible refinamiento que contrarrestara este error podría ser la restricción del rango del atributo DQRS. La baja o nula tasa de falsos positivos en el análisis de éstos registros indica también que el autómata ignoró los trazos anormales de ECG durante el proceso de reconocimiento, mostrando con ello la capacidad que el sistema tiene para discernir entre formas normales y anormales de complejos

Por último debe observarse la Tabla 5, que contiene los resultados de la prueba de vulnerabilidad a los cambios en la muestra de patrones usada para la inferencia. El hecho de que se mantengan los niveles de detección para conjuntos diversos de muestras es un buen indicador de la repetibilidad del método de reconocimiento de patrones diseñado en este trabajo. Destaca sin embargo el 56.15% de detecciones correctas obtenido en el primer caso reportado en la Tabla 5. La causa probable de este porcentaje de

detección tan bajo puede ser principalmente la elección de una de las muestras de entrenamiento, que introdujo al lenguaje la frase $S^+?S^+P^+NP^+S^+?S^+P^+?$, que aparece muy contaminada con errores de codificación.

METODO	PRIMITIVAS	*QRS*	*FP*	*FN*	*%DET*	*DFP*
a	SPN	139	35	48	74.33	0.20
a	MSPN	154	20	33	82.35	0.11
a	MVSPN	145	13	42	77.54	0.08
b	SPN	119	0	68	63.64	0
b	MSPN	133	0	54	71.12	0
b	MVSPN	126	0	61	67.38	0
c	SPN	136	1	51	72.73	0.01
c	MSPN	154	0	33	82.35	0
c	MVSPN	144	0	43	77.01	0
d	SPN	123	32	64	65.78	0.21
d	MSPN	137	16	50	73.26	0.10
d	MVSPN	130	12	57	69.52	0.08

METODO(*forma* de evaluación de los atributos):
a) prueba de aceptación sin validación de atributos
b) prueba de aceptación + validación de ambos atributos
c) prueba de aceptación + validación de AMD únicamente
d) prueba de aceptación + validación de DQRS únicamente

PRIMITIVAS:
M segmento de pendiente cero, precedido de una primitiva P y seguido por una primitiva N
V segmento de pendiente cero, precedido de una primitiva N y seguido por una primitiva P
S segmento de pendiente cero,
P segmento de pendiente mayor que cero
N segmento de pendiente menor que cero

QRS # complejos detectados correctamente
FP falsos positivos
FN falsos negativos
DFP fracción de detecciones erróneas

Tabla 2. Registro I. Resultados de la prueba de funcionalidad.

METODO	PRIMITIVAS	*QRS*	*FP*	*FN*	*%DET*	*DFP*
a	SPN	260	45	24	91.55	0.15
a	MSPN	269	37	15	94.72	0.12
a	MVSPN	252	21	32	88.73	0.08
b	SPN	244	11	40	85.92	0.04
b	MSPN	253	3	31	89.08	0.01
b	MVSPN	238	2	46	83.80	0.01
c	SPN	260	13	24	91.55	0.05
c	MSPN	268	7	16	94.37	0.03
c	MVSPN	253	4	31	89.05	0.02
d	SPN	248	35	36	87.32	0.12
d	MSPN	252	27	32	88.73	0.09
d	MVSPN	238	14	46	83.80	0.06

METODO(forma de evaluación de los atributos):
a) prueba de aceptación sin validación de atributos
b) prueba de aceptación + validación de ambos atributos
c) prueba de aceptación + validación de AMD únicamente
d) prueba de aceptación + validación de DQRS únicamente

PRIMITIVAS:
M segmento de pendiente cero, precedido de una primitiva P y seguido por una primitiva N
V segmento de pendiente cero, precedido de una primitiva N y seguido por una primitiva P
S segmento de pendiente cero,
P segmento de pendiente mayor que cero
N segmento de pendiente menor que cero

QRS # complejos detectados correctamente
FP falsos positivos
FN falsos negativos
DFP fracción de detecciones erróneas

Tabla 3. Registro II. Resultados de la prueba de funcionalidad.

METODO	ARCHIVO	*QRS*	*FP*	*FN*	*%DET*	*DFP*
a	fibrilación	39	0	4	90.70	0
a	VPC tipo 1	42	1	2	95.45	0.02
a	artefactos	27	1	1	96.43	0.04
a	VPC tipo 2	47	0	0	100	0
a	bradicardia	27	0	2	93.10	0
b	fibrilación	39	0	4	90.70	0
b	VPC tipo 1	41	2	3	93.18	0.05
b	artefactos	27	1	1	96.43	0.04
b	VPC tipo 2	47	0	0	100	0
b	bradicardia	27	0	2	93.10	0

METODO(forma de evaluación de los atributos):
a) prueba de aceptación + validación de ambos atributos
b) prueba de aceptación + validación de AMD únicamente

PRIMITIVAS UTILIZADAS:
M segmento de pendiente cero, precedido de una primitiva P y seguido por una primitiva N
S segmento de pendiente cero,
P segmento de pendiente mayor que cero
N segmento de pendiente menor que cero

QRS # complejos detectados correctamente
FP falsos positivos
FN falsos negativos
DFP fracción de detecciones erróneas

Tabla 4. Registros de ECG simulado. Resultados de la prueba de diferenciación de morfologías.

METODO	ARCHIVO	*QRS*	*FP*	*FN*	*%DET*	*DFP*
a	Registro I	105	0	82	56.15	0
a	Registro I	144	0	43	77.01	0
b	Registro I	164	0	23	87.70	0
b	Registro I	138	0	49	73.80	0
a	Registro II	247	1	37	86.97	0
a	Registro II	263	29	21	92.28	
b	Registro II	264	13	20	92.96	
b	Registro II	266	0	18	93.66	0

METODO(forma de evaluación de los atributos):
a) prueba de aceptación + validación de ambos atributos
b) prueba de aceptación + validación de AMD únicamente

PRIMITIVAS UTILIZADAS:
M segmento de pendiente cero, precedido de una primitiva P y seguido por una primitiva N
S segmento de pendiente cero,
P segmento de pendiente mayor que cero
N segmento de pendiente menor que cero

QRS # complejos detectados correctamente
FP falsos positivos
FN falsos negativos
DFP fracción de detecciones erróneas

Tabla 5. Procesamiento de los Registros I y II con diversos autómatas generados con muestras al azar.

CAPITULO 9

CONCLUSIONES

En el presente trabajo se ha descrito una metodología para la detección de complejos QRS normales en registros de electrocardiografía ambulatoria digital, codificados mediante el algoritmo AZTEC, basada en el análisis estructural sintáctico. Aún cuando la aplicación de las técnicas sintácticas al problema de la electrocardiografía ya ha sido estudiada por otros autores, este trabajo ha ofrecido un acercamiento novedoso que se sustenta en la teoría de la inferencia gramatical.

La característica distintiva del método diseñado consiste en que no se realiza ninguna suposición a priori en relación a la clase de patrones que representa a los complejos normales del sujeto que será estudiado, sino que la clase de patrones es "particularizada" al sujeto en cuestión, pues se define experimentalmente a partir de una muestra aleatoria de complejos de su propio registro electrocardiográfico. De esta manera, en cada estudio se cuenta efectivamente con un analizador sintáctico adaptado al problema particular.

Se destaca también el hecho de que el lenguaje utilizado para la descripción de los patrones sea regular, dada su simplicidad de construcción y la muy conveniente relación (desde el punto de vista práctico) que estos lenguajes guardan con los autómatas de estados finitos. Otros autores, presuponiendo que la descripción estructural del ECG requiere de gramáticas con mucha riqueza descriptiva (usualmente de Tipo 1), han eliminado de entrada la posibilidad de utilizar un lenguaje regular. En este trabajo se ha mostrado que es suficiente una gramática regular de tres o cuatro símbolos terminales para describir a un complejo normal en forma eficiente.

Para mejorar las características de aceptación del autómata "individualizado", la gramática regular ha sido extendida con dos atributos, la amplitud máxima diferencial y la duración del complejo. Estos atributos son también calculados al tiempo en que se toma la muestra, de manera que el sistema de detección no depende de constantes determinadas por ensayo, umbrales predefinidos o medidas estadísticas de una población ajena al sujeto que será estudiado. Se ha dimensionado también la utilidad de cada uno de los atributos, destacando la amplitud máxima diferencial como la responsable de abatir la tasa de detecciones positivas erróneas.

Ahora que se ha comprobado la capacidad de detección de complejos normales que manifiesta el uso del autómata (72% de detecciones correctas en promedio), la metodología planteada resulta muy atractiva para aplicarse en sistemas de electrocardiografía ambulatoria digital. El uso de un autómata de estados finitos como base del mecanismo de detección abre la posibilidad de implantar un algoritmo que reconozca complejos normales en línea con la compresión de datos, todo dentro de una plataforma de cómputo elemental contenida en la unidad portátil de registro.

La implantación del algoritmo de inferencia gramatical, así como la del generador de autómatas finitos se ha realizado en forma de una herramienta computacional reutilizable. De esta manera queda allanado el camino para iniciar posteriores aplicaciones que utilicen la estrategia sintáctica.

En perspectiva, el trabajo aquí iniciado tiene varias opciones de continuación. Quizá la más atractiva sea expandir el sistema para que sea capaz de detectar no sólo el complejo QRS normal, sino todo un ciclo normal de ECG. Seguramente se tendría que enriquecer el conjunto de primitivas, corriendo el riesgo, como ya se ha discutido, de hacer representaciones muy específicas que demeriten la capacidad de detección. Una salida interesante podría ser el uso de varios autómatas finitos trabajando concurrentemente

para detectar cada onda específica, colaborando en la prueba de aceptación en forma conjunta.

Otro problema que podría atacarse desde la perspectiva del análisis sintáctico es el de clasificación de arritmias, que es el siguiente paso lógico en el diseño de un sistema completo de análisis del ECG ambulatorio. Dado que las restricciones de la unidad portátil no son aplicables a la tarea de la clasificación (la cual puede ser implantada en una computadora personal) el esquema puede extenderse a lenguajes de mejor capacidad descriptiva. De esta manera, el análisis del ECG se realizaría en dos etapas: una clasificación general mediante la técnica de inferencia desarrollada aquí, que opere dentro de la unidad portátil, seguida de una clasificación especializada efectuada en una computadora de mayor capacidad. Sin embargo, dado que las arritmias son eventos aislados, la aplicación de la inferencia gramatical será restringida, ya que los lenguajes que describan arritmias tendrán que ser diseñados a partir de información conocida previamente. La inferencia gramatical podría usarse entonces para adaptar o mejorar las descripciones originales de las arritmias, con base en la nueva información que aporten en cada análisis las morfologías detectadas. Como se ve, el análisis estructural sintáctico puede todavía aportar herramientas interesantes al análisis automatizado en electrocardiografía.

APENDICE A

```
// Definición de una Gramática Regular

#ifndef _GRAM_REG_
#define _GRAM_REG_

#include <rw/collstr.h>
#include <rw/cstring.h>
#include <rw/regexp.h>
#include <rw/slistcol.h>
#include <rw/stackcol.h>
#include <rw/hashdict.h>
#include <fstream.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include "funtran.h"
#include "tabtrad.h"
#include "inscad.h"

class GramaticaRegular
{
 RWCString                          lenguaje;
 RWCString                          terminales;
 RWSlistCollectables                automata;
 RWSlistCollectables              * muestra;
 RWHashDictionary                 * tabla_subcadenas;
 RWCString                        * caracteres;
 RWCollectableString              * candidato;
 RWCString                        * sustituto;
 RWSlistCollectablesStack         * traductor;
 RWBoolean                          inferible;

 public:

 // Constructores y destructores públicos

 GramaticaRegular(void);
 GramaticaRegular(const char * archivo,unsigned tipo);
 ~GramaticaRegular(void);

 // Funciones miembro públicas

 void                          Infiere(void);
 void                          ConstruyeAutomata(void);
 void                          MuestraAutomata(void);
 void                          ConstruyeLenguaje(void);
 void                          AgregaMuestra(const char * patron);
 RWBoolean                     Prueba(const char * patron);
 RWCString&                    Lenguaje(void);
 RWCString&                    Terminales(void);
```

```
RWBoolean                       Acepta(const char * pat);
RWSlistCollectables *           Automata(void) { return(&automata); }

private:
// Funciones miembro privadas

void                    InicializaVars(void);
void                    ActualizaTablaSubcadenas(RWCString cad);
void                    InsertaSubcadena(RWCSubString subcad,
                                         unsigned reps);
void                    BuscaCandidato(void);
RWCString               CaracterDisponible(void);
void                    ActualizaTablaTraducciones(void);
void                    SustituyeCaracterCandidato(RWCollectableString *
                                                   cad);
void                    ReduceMuestra(void);
void                    ExtraeTerminales(void);
void                    DefineEstadoInicial(FuncionDeTransicion ** edo);
void                    DefineEstadoAceptacion(FuncionDeTransicion *
                                               edo);
void                    DefineEstadoTrampa(FuncionDeTransicion ** edo);
void                    HazAEF(FuncionDeTransicion ** ultimo,
                               unsigned * resto, unsigned limite,
                               FuncionDeTransicion ** ret_edo,
                               unsigned * ret_ent,unsigned primero );
void                    ResuelveNoDets(FuncionDeTransicion ** ultimo);
RWBoolean               NoDetResuelta(RWSlistCollectables res,
                                      RWSlistCollectables nd);
void                    TransfiereTransiciones(FuncionDeTransicion *edo,
                                               RWSlistCollectables *
                                               trans);
void                    SustituyeTransiciones(RWSlistCollectables
                                              res,unsigned ult);
void                    MarcaEstadosAccesibles(FuncionDeTransicion* edo);
void                    ReducePorAcceso(void);
void                    CompletaAEF(FuncionDeTransicion * trampa);
void                    MensajeError(unsigned err);
};

// Macros diversos

#define  CARACTERES_INICIALES       "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
#define  MAXPATRON                  256
#define  ARCHIVO                    0
#define  LENGUAJE                   1
#define  HayTablaSubcadenas()       !(*tabla_subcadenas).isEmpty()
#define  LimpiaTablaSubcadenas()
         (*tabla_subcadenas).clearAndDestroy()

// #define   _GR_DEBUG_

#define  CTRL_DESP              20
#define  DEBUG(a,b)             cout << NL << "DEBUG >>> " << a << b << NL
#define  NO_ARCH                0
#define  SINTAXIS               1
#define  TIPO_CONST             2
#define  MEM_MUESTRA            3
#define  MEM_SUBCADENA          4
#define  MEM_TRAD               5
#define  MEM_ESTADO             6
```

```
#define  MEM_CONST              7

extern                          char GRERR[8][100];

#endif

#include "gramreg.h"

char GRERR[8][100] = {
        "Error abriendo archivo", "Error de sintaxis en el lenguaje",
        "Tipo de constructor desconocido",
        "Memoria insuficiente para otra muestra",
        "Memoria insuficiente para obtener subcadenas",
        "Memoria insuficiente para definir traducciones",
        "Memoria insuficiente para incluir otro  estado"
        };
```

```
void GramaticaRegular::InicializaVars(void)
{
 caracteres = new RWCString(CARACTERES_INICIALES);
 if(caracteres)
 {
  muestra = new RWSlistCollectables;
  if(muestra)
  {
   tabla_subcadenas = new RWHashDictionary;
   if(tabla_subcadenas)
   {
    traductor = new RWSlistCollectablesStack;
    if(traductor)
    {
     sustituto = new RWCString;
     if(!sustituto)
     {
      delete traductor;
      delete tabla_subcadenas;
      delete muestra;
      delete caracteres;
     }
    }
    else
    {
     delete tabla_subcadenas;
     delete muestra;
     delete caracteres;
    }
   }
   else
   {
    delete muestra;
    delete caracteres;
   }
  }
  else
   delete caracteres;
 }
 if(!caracteres)
  MensajeError(MEM_CONST);
 else
 {
  candidato = NULL;
  inferible = TRUE;
 }
}
```

```
GramaticaRegular::GramaticaRegular(void)
{
 InicializaVars();
}
```

```
GramaticaRegular::~GramaticaRegular(void)
{
 FuncionDeTransicion * tmp_e;
 RWCollectableString * tmp_s;

 #ifdef _GR_DEBUG_
 DEBUG("Memoria libre (antes de eliminar) : ",coreleft());
 getch();
 #endif
 automata.clearAndDestroy();
 (*traductor).clearAndDestroy();
 delete caracteres;
 delete sustituto;
 delete candidato;
 #ifdef _GR_DEBUG_
 DEBUG("Memoria libre (luego de eliminar) : ",coreleft());
 getch();
 #endif
}
```

```
GramaticaRegular::GramaticaRegular(const char * archivo,unsigned tipo)
{
 ifstream arch(archivo);
 char patron[MAXPATRON];
 RWCString tmp(archivo);
 RWBoolean abierto;
 unsigned i = 0, par_abre = 0, par_cierra = 0, suma = 0;

 if(tipo == ARCHIVO)
 {
  if(arch)
  {
   InicializaVars();
   while(arch.getline(patron,MAXPATRON))
   {
    AgregaMuestra(patron);
    #ifdef _GR_DEBUG_
    DEBUG("Lectura de archivo. Patrón ",i++);
    cout << patron;
    getch();
    #endif
   }
   Infiere();
  }
  else
   MensajeError(NO_ARCH);
 }
 else if(tipo == LENGUAJE)
       {
        if(tmp[tmp.length() - 1] != '/')
         tmp += "/";
for(i = 0; i < tmp.length() ; i++) switch(tmp[i])
         {
          case '(' :par_abre++;
                    abierto = TRUE;
break;
case ')' : par_cierra++; if(abierto)
MensajeError(SINTAXIS);
break;
          case '+' : suma++;
if((i == 0) || (tmp[i-1] != ')')) MensajeError(SINTAXIS);
                    break;
          default  : abierto =FALSE;
         }
if((par_abre != par_cierra) ||
    (par_abre != suma)) MensajeError(SINTAXIS);
       lenguaje = tmp;
       inferible = FALSE;
      }
      else
       MensajeError(TIPO_CONST);
}
```

```
void GramaticaRegular::Infiere(void)
{
 RWBoolean otro_paso;
 #ifdef _GR_DEBUG_
 unsigned cta = 0;
 #endif

 if(inferible)
  do
  {
   RWCollectableString * cadena;
   RWSlistCollectablesIterator mi(*muestra);

while((cadena = (RWCollectableString *)mi()) != NULL)
  ActualizaTablaSubcadenas(*cadena); if(HayTablaSubcadenas())
   {
    BuscaCandidato();
    ActualizaTablaTraducciones();
    mi.reset();
while((cadena = (RWCollectableString *)mi()) != NULL)
    SustituyeCaracterCandidato(cadena);
    LimpiaTablaSubcadenas();
    candidato = NULL;
    otro_paso = TRUE;
    #ifdef _GR_DEBUG_
DEBUG("Avance en la inferencia. Paso ",cta++);
    mi.reset();
while((cadena = (RWCollectableString *)mi()) != NULL)
 cout << (*cadena) << NL;
 getch();
    #endif
   }
   else otro_paso = FALSE;
  }
  while(otro_paso);
 ReduceMuestra();
 ConstruyeLenguaje();
}
```

```
void GramaticaRegular::ReduceMuestra(void)
{
 RWCollectableString * ant, * post;
 unsigned i,j;

 for(i = 0 ; i < (*muestra).entries() - 1 ; i++)
 {
  ant = (RWCollectableString *)(*muestra).at(i);
  for(j = i + 1 ; j < (*muestra).entries() ; j++)
  {
   post = (RWCollectableString *)(*muestra).at(j);
   if((*ant).isEqual(post))
   {
    (*muestra).removeAndDestroy(post);
    j--;
   }
  }
 }
}
```

```
void GramaticaRegular::ConstruyeLenguaje(void)
{
 TablaTraduccion * trad;
 RWSlistCollectablesIterator mi(*muestra);
 RWCollectableString * cad;
 RWCString nula;

 while((trad = (TablaTraduccion *)traductor->pop()) != NULL)
 {
  RWCString cambio((trad->Subcadena()) += ")+");
  cambio.prepend("(");
  RWRegexp act(trad->Sustituto());
  mi.reset();
  while((cad = (RWCollectableString *)mi()) != NULL)
   while((*cad)(act) != nula)
    (*cad)(act) = cambio;
 }
 mi.reset();
 while((cad = (RWCollectableString *)mi()) != NULL)
  lenguaje += ((*cad) += "/");
 #ifdef _GR_DEBUG_
 DEBUG("Lenguaje inferido: ",lenguaje);
 getch();
 #endif
}
```

```
void GramaticaRegular::ConstruyeAutomata(void)
{
 int rango;
 unsigned indice,estado,empezar;
 FuncionDeTransicion * ultimo, * temp, * inicial;

 if(lenguaje != "")
 {
  ExtraeTerminales();
  DefineEstadoInicial(&inicial);
  indice = 0;
  empezar = 1;
  while((rango = lenguaje.index("/",indice)) != -1)
  {
   ultimo = inicial;
   HazAEF(&ultimo,&indice,rango,&temp,&estado,empezar);
   DefineEstadoAceptacion(ultimo);
   indice = rango + 1;
   empezar = (*ultimo).Estado() + 1;
  }
  #ifdef _GR_DEBUG_
  DEBUG("Automata incompleto [no deterministico] : ","");
  MuestraAutomata();
  getch();
  #endif
  ResuelveNoDets(&ultimo);
  DefineEstadoTrampa(&ultimo);
  CompletaAEF(ultimo);
  #ifdef _GR_DEBUG_
  DEBUG("Automata deterministico completo : ","");
  MuestraAutomata();
  getch();
  #endif
  MarcaEstadosAccesibles(inicial);
  ReducePorAcceso();
  #ifdef _GR_DEBUG_
  DEBUG("Reducción de estados inaccesibles : ","");
  MuestraAutomata();
  getch();
  #endif
 }
}
```

```
void GramaticaRegular::MuestraAutomata(void)
{
 FuncionDeTransicion * tmp;
 unsigned i = 0;

 RWSlistCollectablesIterator ti(automata);
 while((tmp = (FuncionDeTransicion *)ti()) != NULL)
 {
  tmp->EscribeTransicion(terminales);
  i++;
  if(i == CTRL_DESP)
  {
   getch();
   i = 0;}
 }
}
```

```
void GramaticaRegular::AgregaMuestra(const char * patron)
{
 unsigned i = 0;
 RWCollectableString * muestra_nueva;

 if(inferible)
 {
  RWCString prueba(patron);
  for(i = 0 ; i < prueba.length() ; i++)
  {
   RWRegexp quita(RWCString(prueba(i,1)));
   (*caracteres)(quita) = "";
  }
  muestra_nueva = new RWCollectableString(patron);
  if(muestra_nueva)
   (*muestra).insert(muestra_nueva);
  else
   MensajeError(MEM_MUESTRA);
 }
}
```

```
void GramaticaRegular::ActualizaTablaSubcadenas(RWCString cad)
{
 unsigned        long_cad,long_subcad;
 unsigned        pos_piv,pos_loc;
 unsigned        rango_piv,rango_loc,rango_subcad;
 unsigned        reps;

 long_cad = cad.length();
 rango_subcad = long_cad/2;
 for(long_subcad = 1 ; long_subcad <= rango_subcad ; long_subcad++)
 {
  pos_piv = 0;
  pos_loc = long_subcad;
  reps = 0;
  rango_piv = long_cad - 2*long_subcad;
  rango_loc = long_cad - long_subcad;
  while(pos_piv <= rango_piv)
  {
   while(pos_loc <= rango_loc)
   {
 if(cad(pos_piv,long_subcad) == cad(pos_loc,long_subcad)) {
      reps ++;
      pos_loc += long_subcad;
     }
     else break;
   }
   if(reps)
   {
    InsertaSubcadena(cad(pos_piv,long_subcad),reps);
    pos_piv += long_subcad*reps;
   }
   pos_piv ++;
   pos_loc = pos_piv + long_subcad;
   reps = 0;
  }
 }
}
```

```
RWCString& GramaticaRegular::Lenguaje(void)
{
 return(lenguaje);
}
```

```
RWCString& GramaticaRegular::Terminales(void)
{
 return(terminales);
}
```

```
void   GramaticaRegular::InsertaSubcadena(RWCSubString   subcad,unsigned
reps)
{
 RWCollectableString * temp_str;
 InfoSubcad * temp_inf, * temp_inf_mod;
 temp_str = new RWCollectableString(subcad);
 if(temp_str)
 {
  temp_inf = new InfoSubcad;
  if(temp_inf)
  {
   *temp_inf = reps;
    temp_inf_mod=(InfoSubcad*)((*tabla_subcadenas).findValue(temp_str));
   if(temp_inf_mod)
    *temp_inf_mod = reps;
   else
    (*tabla_subcadenas).insertKeyAndValue(temp_str,temp_inf);
  }
  else
   delete temp_str;
 }
 if(!temp_str)
  MensajeError(MEM_SUBCADENA);
}
```

```
RWCString GramaticaRegular::CaracterDisponible(void)
{
 RWCString tmp((*caracteres)(0,1));
 (*caracteres)(0,1) = "";
 return(tmp);
}
```

```
void GramaticaRegular::BuscaCandidato(void)
{
 RWHashDictionaryIterator bc(*tabla_subcadenas);
 RWCollectableString * llave;
 InfoSubcad * valor;
 InfoSubcad * act_inf;

 while((llave = (RWCollectableString *)bc()) != NULL)
 {
  if(candidato == NULL)
   candidato = llave;
  else
  {
 act_inf = (InfoSubcad *)tabla_subcadenas->findValue(candidato);
 valor = (InfoSubcad *)tabla_subcadenas->findValue(llave);
 if(valor->Reps() > act_inf->Reps())
     candidato = llave;
    else if ((valor->Reps() == act_inf->Reps()) &&
               (valor->Aps() > act_inf->Aps()))
          candidato = llave;
  }
 }
}
```

```
void GramaticaRegular::ActualizaTablaTraducciones(void)
{
 TablaTraduccion * tmp;

 *sustituto = CaracterDisponible();
 tmp = new TablaTraduccion(*candidato,*sustituto);
 if(tmp)
  (*traductor).push(tmp);
 else
  MensajeError(MEM_TRAD);
}
```

```
void GramaticaRegular::DefineEstadoInicial(FuncionDeTransicion ** edo)
{
 *edo = new FuncionDeTransicion(0,terminales.length(),FTINICIAL);
 if(*edo)
  automata.insert(*edo);
 else
  MensajeError(MEM_ESTADO);
}
```

```
void GramaticaRegular::SustituyeCaracterCandidato(RWCollectableString *
cad) {
 RWCString nula;
 unsigned i = 0;

 if(candidato->length() == 1)
 {
  RWRegexp sust((*candidato) += "+");
  while((*cad)(sust) != nula)
   (*cad)(sust) = *sustituto;
 }
 else
 {
  RWRegexp sust_tmp(*candidato);
  RWCString cambio(*sustituto);
  while((*cad)(sust_tmp) != nula)
   (*cad)(sust_tmp) = *sustituto;
  RWRegexp sust(cambio += "+");
  while((*cad)(sust,i++) != nula)
   (*cad)(sust) = *sustituto;
 }
}
```

```
void GramaticaRegular::ExtraeTerminales(void)
{
 RWRegexp  ops("[(/\+)]");
 char      tmp;
 unsigned  i,j;

 terminales = lenguaje;
 while(terminales(ops) != "")
 terminales(ops) = "";
 for(i = 0 ; i < terminales.length()  - 1 ; i++)
  for(j = i + 1 ; j < terminales.length() ; j++)
   if(terminales(i,1) == terminales(j,1))
    terminales(j--,1) = "";
 for(i = 0 ; i < terminales.length()  - 1 ; i++)
  for(j = i ; j < terminales.length() ; j++)
   if(terminales[i] > terminales[j])
   {
    tmp = terminales[i];
    terminales[i] = terminales[j]; terminales[j] = tmp;
   }
 #ifdef _GR_DEBUG_
 DEBUG("Símbolos terminales : ",terminales);
 getch();
 #endif
}
```

```
void GramaticaRegular::DefineEstadoAceptacion(FuncionDeTransicion * edo)
{
 (*edo).DefineTipo(FTACEPTACION);
}
```

```
void GramaticaRegular::DefineEstadoTrampa(FuncionDeTransicion ** edo)
{
 FuncionDeTransicion * tmp;

tmp = new FuncionDeTransicion((*(*edo)).Estado() + 1,
                              terminales.length(),FTTRAMPA);
 if(tmp)
 {
  automata.insert(tmp);
  *edo = tmp;
 }
 else
  MensajeError(MEM_ESTADO);
}
```

```
void GramaticaRegular::SustituyeTransiciones(RWSlistCollectables res,
                                             unsigned ult)
{
 RWSlistCollectablesIterator lit(res);
 RWSlistCollectables * tmp_l, * tmp_t;
 FuncionDeTransicion * tmp_e;
 int i,j;

 ult++;
 while((tmp_l = (RWSlistCollectables *)lit()) != NULL)
 {
  for(j = automata.entries() - 1 ; j >= 0 ; j--)
  {
   tmp_e = (FuncionDeTransicion *)automata.at(j);
   for(i = 0 ; i < terminales.length() ; i++)
   {
    tmp_t = (*tmp_e).ObtenListaTransiciones(i);
    if((*tmp_t) == (*tmp_l)) {
     (*tmp_t).clear();
     (*tmp_t).insert(automata.at(ult));}
   }
  }
  ult++;}
}
```

```
RWBoolean GramaticaRegular::NoDetResuelta(RWSlistCollectables res,
                                          RWSlistCollectables nd)
{
 RWSlistCollectables * tmp;

 RWSlistCollectablesIterator rit(res);
 while((tmp = (RWSlistCollectables *)rit()) != NULL)
   if((*tmp) == nd)
    return(TRUE);
 return(FALSE);
}
```

```
void GramaticaRegular::TransfiereTransiciones(FuncionDeTransicion * edo,
                                              RWSlistCollectables*trans)
{
 RWSlistCollectablesIterator tit(*trans);
 RWSlistCollectables * lista_trans;
 FuncionDeTransicion * edo_tmp, * edo_trans;
 unsigned i;

 while((edo_tmp = (FuncionDeTransicion *)tit()) != NULL)
 {
  for(i = 0 ; i < terminales.length() ; i++)
  {
   lista_trans = (*edo_tmp).ObtenListaTransiciones(i);
   if(lista_trans)
   {
    RWSlistCollectablesIterator tit2(*lista_trans);
    while((edo_trans = (FuncionDeTransicion *)tit2()) != NULL)
     (*edo).InsertaTransicion(edo_trans,i);
   }
  }
  if((*edo_tmp).Tipo() == FTACEPTACION)
   (*edo).DefineTipo(FTACEPTACION);
 }
}
```

```
void GramaticaRegular::HazAEF(FuncionDeTransicion ** ultimo,
                              unsigned * resto, unsigned limite,
                              FuncionDeTransicion ** ret_edo,
                              unsigned * ret_ent,unsigned primero)
{
 FuncionDeTransicion * n_estado, * prim_edo;
 unsigned                    prim_ent;
 RWBoolean                   define_primero;
 char                        token;

 define_primero = TRUE;
 while((*resto) < limite)
  switch((token = lenguaje[*resto]))
  {
   case '(' : {
               (*resto)++;
               if(define_primero)
               {
                HazAEF(ultimo,resto,limite,ret_edo,ret_ent,primero);
                prim_edo = *ret_edo;
                prim_ent = *ret_ent;
                define_primero = FALSE;
               }
               else
                HazAEF(ultimo,resto,limite,ret_edo,ret_ent,
                       (*(*ultimo)).Estado() + 1);
               break;
              }
   case ')' : {
               (*(*ultimo)).InsertaTransicion(prim_edo,prim_ent);
               (*resto)+=2;
               *ret_edo = prim_edo;
               *ret_ent = prim_ent;
               return;
              }
   default  : {
               if(define_primero)
                n_estado = new FuncionDeTransicion(primero,
                                   terminales.length(),FTTRANS);
               else
                n_estado = new
                            FuncionDeTransicion((*(*ultimo)).Estado()+1,
                                   terminales.length(),FTTRANS);
               if(n_estado)
               {
                automata.insert(n_estado);
                (*(*ultimo)).InsertaTransicion(n_estado,
                                   terminales.first(token));
                if(define_primero)
                {
                 prim_edo = n_estado;
                 prim_ent = terminales.first(token);
                 define_primero = FALSE;
                }
                *ultimo = n_estado;
                (*resto)++;}
               else
                MensajeError(MEM_ESTADO);}}}
```

```
void GramaticaRegular::ResuelveNoDets(FuncionDeTransicion ** ultimo)
{
 RWSlistCollectables listas;
 RWSlistCollectables * trans_por_term, * tmp;
 FuncionDeTransicion * edo, * nuevo_edo;
 unsigned i,j,ult;
 #ifdef _GR_DEBUG_
 unsigned cta = 0;
 #endif

 ult = (*(*ultimo)).Estado();
 for(j = 0 ; j < automata.entries() ; j++)
 {
  edo = (FuncionDeTransicion *)automata.at(j);
  for(i = 0 ; i < terminales.length() ; i++)
  {
   trans_por_term = (*edo).ObtenListaTransiciones(i);
   if((trans_por_term) && ((*trans_por_term).entries() > 1))
   {
    if(!NoDetResuelta(listas,*trans_por_term))
    {
     listas.insert(trans_por_term);
     nuevo_edo = new FuncionDeTransicion((*(*ultimo)).Estado() + 1,
                                         terminales.length(), FTTRANS);
     if(nuevo_edo)
     {
      automata.insert(nuevo_edo);
      *ultimo = nuevo_edo;
      TransfiereTransiciones(nuevo_edo,trans_por_term);
      #ifdef _GR_DEBUG_
      DEBUG("Reduccion del automata. Paso ",cta++);
      DEBUG("Estados totales : ",automata.entries());
      MuestraAutomata();
      getch();
      #endif
     }
     else
      MensajeError(MEM_ESTADO);
    }
   }
  }
 }
 SustituyeTransiciones(listas,ult);
 #ifdef _GR_DEBUG_
 DEBUG("Redefinición de transiciones no determinísticas","");
 MuestraAutomata();
 getch();
 #endif
}
```

```
void GramaticaRegular::MarcaEstadosAccesibles(FuncionDeTransicion *
                                              edo)
{
 unsigned i;

 if(!(*edo).Accesible())
 {
  (*edo).MarcaAccesible();
  for(i=0;i<terminales.length();i++)
  MarcaEstadosAccesibles((*edo).ObtenTransicion(i));
 }
}
```

```
void GramaticaRegular::CompletaAEF(FuncionDeTransicion * trampa)
{
 RWSlistCollectablesIterator eit(automata);
 FuncionDeTransicion * edo;

 while((edo = (FuncionDeTransicion *)eit()) != NULL)
  (*edo).CompletaTransicion(trampa);
}
```

```
void GramaticaRegular::ReducePorAcceso(void)
{
 FuncionDeTransicion * tmp;
 unsigned i;

 for(i = 0 ; i < automata.entries() ; i++)
 {
  tmp = (FuncionDeTransicion *)automata.at(i);
  if(!(*tmp).Accesible())
  {
   automata.remove(tmp);
   i--;
  }
 }
}
```

```
RWBoolean GramaticaRegular::Acepta(const char * pat)
{
 RWCString patron(pat);
 RWCString p("[^");
 FuncionDeTransicion * tmp;

 unsigned i;

 p += terminales;
 p += "]";
 RWRegexp prueba(p);
 #ifdef _GR_DEBUG_
 DEBUG("Sintaxis ",(patron(prueba) == "" ? "OK" : "errónea"));
 getch();
 #endif
 if(patron(prueba) == "")
 {
  tmp = (FuncionDeTransicion *)automata.first();
  for(i = 0 ; i < patron.length() ; i++)
  {
 tmp=(*tmp).ObtenTransicion(terminales.first(patron[i]));
  if((*tmp).Tipo() == FTTRAMPA)
     return(FALSE);
  }
  if((*tmp).Tipo() == FTACEPTACION)
   return(TRUE);
  else
   return(FALSE);
 }
 return(FALSE);
}
```

```
void GramaticaRegular::MensajeError(unsigned err)
{
 cout << "ERROR >>> " << GRERR[err] << NL;
 cout << "-- Programa terminado --";
 exit(err);
}
```

```
// Auxiliar para tabla de subcadenas

#ifndef _INFO_SUBCAD_
#define _INFO_SUBCAD_
#define __INFOSUBCAD        0x4000

#include <rw/collect.h>

class InfoSubcad : public RWCollectable
{
 RWDECLARE_COLLECTABLE(InfoSubcad)

public:

// Constructores públicos

 InfoSubcad(void);
 ~InfoSubcad(void);

// Funciones miembro públicas

 int                          compareTo(const RWCollectable * inf_sub)
                                        const;
 RWBoolean                    isEqual(const RWCollectable * inf_sub)
                                        const;
 unsigned                     hash(void) const;
 unsigned                     Reps(void);
 unsigned                     Aps(void);

// Operadores públicos
 InfoSubcad& operator = (unsigned rep);

// Variables privadas
 private:
 unsigned max_reps;
 unsigned num_aps;

};


#include "inscad.h"
```

```
InfoSubcad::InfoSubcad(void)
{
 max_reps = 0;
 num_aps = 0;
}
```

```
InfoSubcad::~InfoSubcad(void)
{
}

RWDEFINE_COLLECTABLE(InfoSubcad,__INFOSUBCAD)
```

```
int InfoSubcad::compareTo(const RWCollectable * inf_sub) const
{
 const InfoSubcad * temp;

 temp = (const InfoSubcad *)inf_sub;
 if(max_reps == temp->max_reps)
  if(num_aps == temp->num_aps)
   return(0);
  else return(num_aps > temp->num_aps ? 1 : -1);
 else return(max_reps > temp->max_reps ? 1 : -1);

 }
```

```
RWBoolean InfoSubcad::isEqual(const RWCollectable * inf_sub) const
{
 const InfoSubcad * temp;
 temp = (const InfoSubcad *)inf_sub;
 if((max_reps == temp->max_reps) && (num_aps == temp->num_aps))
  return(TRUE);
 else return(FALSE);
}
```

```
unsigned InfoSubcad::hash(void) const
{
 return(max_reps);
}
```

```
unsigned InfoSubcad::Reps(void)
{
 return(max_reps);
}
```

```
unsigned InfoSubcad::Aps(void)
{
 return(num_aps);
}
```

```
InfoSubcad& InfoSubcad::operator = (unsigned rep)
{
 if(rep > max_reps)
  max_reps = rep;
 num_aps++;
 return(*this);
}
```

```
// Definición de una tabla de traduccion

#ifndef _TAB_TRAD_
#define _TAB_TRAD_
#define __TABLATRADUCCION       0x4001

#include <rw/collect.h>
#include <rw/cstring.h>

class TablaTraduccion : public RWCollectable
{
 RWDECLARE_COLLECTABLE(TablaTraduccion)

 public:

// Constructores públicos

 TablaTraduccion(void);
 TablaTraduccion(RWCString sc,RWCString st);


 ~TablaTraduccion(void);

// Funciones miembro públicas

 int                          compareTo(const RWCollectable * tab) const;
 RWBoolean                    isEqual(const RWCollectable * tab) const;
 unsigned                     hash(void) const;
 RWCString                    Subcadena(void);
 RWCString                    Sustituto(void);

// Variables privadas

 private:

 RWCString                    subcad;
 RWCString                    sust;
};


#include "tabtrad.h"
```

```
TablaTraduccion::TablaTraduccion(void)
{
}
```

```
TablaTraduccion::TablaTraduccion(RWCString sc,RWCString st)
{
 subcad = sc;
 sust = st;
}
```

```
TablaTraduccion::~TablaTraduccion(void)
{
}
```

```
RWDEFINE_COLLECTABLE(TablaTraduccion,__TABLATRADUCCION)

int TablaTraduccion::compareTo(const RWCollectable * tab) const
{
 const TablaTraduccion * temp;
 temp = (const TablaTraduccion *)tab;
 if(temp->subcad == subcad)
   return(0);
 else return(temp->subcad > subcad ? 1 : -1);
}
```

```
RWBoolean TablaTraduccion::isEqual(const RWCollectable * tab) const
{
 const TablaTraduccion * temp;
 temp = (const TablaTraduccion *)tab;
 if((subcad == temp->subcad) && (sust == temp->sust)) return(TRUE);
 else return(FALSE);
}
```

```
unsigned TablaTraduccion::hash(void) const
{
 return(subcad.hash());
}
```

```
RWCString TablaTraduccion::Subcadena(void)
{
 return(subcad);
}
```

```
RWCString TablaTraduccion::Sustituto(void)
{
 return(sust);
}
```

```
// Definicion de una funcion de transicion

#ifndef _FUNC_TRANS_
#define _FUNC_TRANS_
#define __FUNCIONTRANS  0x4002

#include <rw/collect.h>
#include <rw/slistcol.h>
#include <rw/ordcltn.h>
#include <rw/rstream.h>
#include <rw/cstring.h>
#include <iomanip.h>

class FuncionDeTransicion : public RWCollectable
{
 RWDECLARE_COLLECTABLE(FuncionDeTransicion)

 public:

// Constructores públicos

 FuncionDeTransicion(void);
 FuncionDeTransicion(unsigned edo,unsigned num_ents,unsigned tipo);
 ~FuncionDeTransicion(void);

// Funciones miembro públicas

int                          compareTo(const RWCollectable * inf_sub)
                                const;
RWBoolean                    isEqual(const RWCollectable * inf_sub) const;
unsigned                     hash(void) const;
void                         InsertaTransicion(FuncionDeTransicion *
                                edo,unsigned ent);
void                         CompletaTransicion(FuncionDeTransicion*
                                trampa);
FuncionDeTransicion *        ObtenTransicion(unsigned ent);
RWSlistCollectables *        ObtenListaTransiciones( unsigned ent);
void                         EscribeTransicion(RWCString ents);
unsigned                     Estado(void);
unsigned                     Tipo(void);
void                         DefineTipo(unsigned tipo);
RWBoolean                    Accesible(void);
void                         arcaAccesible(void);

// Variables privadas

 private:

unsigned estado;
unsigned tipo_estado;
unsigned condicion;
unsigned numero_entradas;
RWOrdered transiciones;
};
```

```
// Macros diversos

#define FTINICIAL                                    1
#define FTTRANS                                      2
#define FTACEPTACION                                 3
#define FTTRAMPA                                     4

#define FTACCESIBLE                                  1
#define FTINACCESIBLE                                2


#include "funtran.h"
```

```
FuncionDeTransicion::FuncionDeTransicion(void) {
 estado = numero_entradas = 0;
 tipo_estado = FTTRANS;
 condicion = FTINACCESIBLE;
}
```

```
FuncionDeTransicion::FuncionDeTransicion(unsigned edo,
                    unsigned num_ents,unsigned tipo)
{
 unsigned i;
 RWSlistCollectables * tmp;
 estado = edo;
 numero_entradas = num_ents;
 tipo_estado = tipo;
 condicion = FTINACCESIBLE;
 transiciones = new RWOrdered;
 for(i = 0 ; i < numero_entradas ; i++)
 {
  tmp = new RWSlistCollectables;
  *(transiciones).insert(tmp);
 }

}
```

```
FuncionDeTransicion::~FuncionDeTransicion(void)
{
unsigned i;

 for(i = 0 ; i < numero_entradas ; i++)
   (*((RWSlistCollectable*)(*transiciones)[i])).clear();

   (*transiciones).clearAndDestroy();

   delete transiciones;

}
```

```
RWDEFINE_COLLECTABLE(FuncionDeTransicion,__FUNCIONTRANS)

int FuncionDeTransicion::compareTo(const RWCollectable * fun_tran)
                          const
{
 FuncionDeTransicion * temp;
 temp = (FuncionDeTransicion *)fun_tran;
 return(estado == temp->estado ? 0 :(estado > temp->estado ? 1 :-1));
}
```

```
RWBoolean FuncionDeTransicion::isEqual(const RWCollectable * fun_tran)
                            const
{
 FuncionDeTransicion * temp;
 temp = (FuncionDeTransicion *)fun_tran;
 return(estado == temp->estado ? TRUE : FALSE);
}
```

```
unsigned FuncionDeTransicion::hash(void) const
{
 return(estado);
}
```

```
void FuncionDeTransicion::InsertaTransicion(FuncionDeTransicion *
edo,unsigned ent) {
 RWSlistCollectables * temp;
 if(edo)
 {
  temp = (RWSlistCollectables *)(*transiciones)[ent];
  if(!(*temp).containsReference(edo))
   (*temp).insert(edo);
 }
}
```

```
void FuncionDeTransicion::CompletaTransicion(FuncionDeTransicion *
                              trampa)
{
 unsigned i;
 RWSlistCollectables * tmp;

 for(i = 0 ; i < numero_entradas ; i++)
 {
  tmp = (RWSlistCollectables *)(*transiciones)[i];
  if((*tmp).isEmpty())
    (*tmp).insert(trampa);
 }
}
```

```
FuncionDeTransicion * FuncionDeTransicion::ObtenTransicion(unsigned ent)
{
 RWSlistCollectables * tmp;

 tmp = (RWSlistCollectables *)(*transiciones)[ent];
 return((FuncionDeTransicion *)(*tmp).first());
}
```

```
RWSlistCollectables *
FuncionDeTransicion::ObtenListaTransiciones(unsigned ent)
{
 return((RWSlistCollectables *)(*transiciones)[ent]);
}
```

```
void FuncionDeTransicion::EscribeTransicion(RWCString ents)
{
 unsigned i,j;
 FuncionDeTransicion * tmp;
 char   t[4][12] = {
                        "INICIAL    ",
                        "NORMAL     ",
                        "ACEPTACION ",
                        "TRAMPA     "
                       };
cout << "Tipo: " << t[tipo_estado - 1] << "Estado: " << setw(3) <<
estado;
 for(i = 0 ; i < numero_entradas ; i++)
 {
  RWSlistCollectablesIterator ti(*(RWSlistCollectables *)
                                  (*transiciones)[i]);
   cout << "  " << ents[i] << ":";
   j = 3;
   while((tmp = (FuncionDeTransicion *)ti()) != NULL)
   {
    cout << " " << setw(3) << tmp->estado;
    j += 4;
   }
   cout << setw((50/numero_entradas) - j) << "";
 }
 cout << NL;
}
```

```
unsigned FuncionDeTransicion::Estado(void)
{
 return(estado);
}
```

```
unsigned FuncionDeTransicion::Tipo(void)
{
 return(tipo_estado);
}
```

```
void FuncionDeTransicion::DefineTipo(unsigned tipo)
{
 if((tipo == FTTRANS) || (tipo == FTINICIAL) ||
       (tipo == FTACEPTACION) || (tipo == FTTRAMPA))
   tipo_estado = tipo;
}
```

```
RWBoolean FuncionDeTransicion::Accesible(void)
{
 if(condicion == FTACCESIBLE)
   return(TRUE);
 else
   return(FALSE);
}
```

```
void FuncionDeTransicion::MarcaAccesible(void)
{
 condicion = FTACCESIBLE;
}
```

REFERENCIAS

[1] Aho A.V., Ullman J.D. "The Theory of Parsing, Translation and Compiling". Prentice Hall, 1972.

[2] Albus J.E. *Electrocardiogram Interpretation Using a Stochastic Finite State Model.* en "Syntactic Pattern Recognition Applications" Fu K.S. ed. Springer Verlag,1976. p.51-64

[3] Armington R.M., Graboys T.B., Lown B., Lenson R. *Semiautomated Data Reduction of Ventricular Ectopic Activity: Methodology and Clinical Application.* Med Inst, 1978. 12:340-342

[4] Belfonte G., DeMori R., Ferraris,F. *A Contribution to the Automatic Processing of Electrocardiograms using Syntactic Methods.* IEEE Trans Biomed Eng, 1979. 26:125-136

[5] Birman K.P. *Rule-Based Learning for More Accurate ECG Analysis.* IEEE Trans Pat Anal Mach Intell, 1982. 4:369-380

[6] Bunke H., Sanfeliu A. eds. "Syntactic and Structural Pattern Recognition. Theory and Applications". World Scientific Publishing Co., 1990

[7] Cashman P.M.M. *The Use of R-R Interval and Difference Histogram in Classifying Disorders of Sinus Rhythm*. J Med Eng Technol, 1977. :20-27

[8] Cox J.R., Nolle F.M., Fozzard H.A., Oliver G.C. *AZTEC: A Preprocessing Program for Real Time ECG Analysis*. IEEE Trans Biomed Eng, 1968. 15:128-129

[9] Escalona O., Passariello G., Mora F. *An Algorithm for Microprocessor-Based QRS Detection*. J Clin Eng, 1986. 11:213-219

[10] Fancott T., Wong D.H. *A Minicomputer System for Direct High Speed Analysis of Cardiac Arrhythmia in 24h Ambulatory ECG Tape Recordings*. IEEE Trans Biomed Eng, 1980. 27:685-693

[11] Feezor M.D., Wallace A.G., Stacy R.W. *A Real Time Waveform Analyzer for Detection of Ventricular Premature Beats*. J Appl Phisiol, 1969. 29:541-545

[12] Friesen G.M., Jannett T.C., Jadallah M.A., Yates S.L., Quint S.R., Nagle H.T. *A Comparison of the Noise Sensitivity of Nine QRS Detection Algorithms*. IEEE trans Biomed Eng, 1990. 37:85-98

[13] Fu K.S., Booth T.L. *Grammatical Inference: Introduction and Survey-Part I*. IEEE Trans Sys Man Cyber, 1975. 5:95-111

[14] Fu K.S., Booth T.L. *Grammatical Inference: Introduction and Survey-Part II*. IEEE Trans Sys Man Cyber, 1975. 5:409-423

[15] González R.C., Thomason M.G. "Syntactic Pattern Recognition". Addison Wesley, 1978.

[16] Guest Editorials. Med Inst, 1978. 12:316-319

[17] Handelsman H. *Real-Time Cardiac Monitors*. Med Elec, 1990. 9:95-101

[18] Holter, N.J. *New Method for Heart Studies: Continuous Electrocardiography of Active Subjects*. Science, 1961. 134:1214-1220

[19] Horowitz S.L. *A Syntactic Algorithm for Peak Detection in Waveforms with Applications to Cardiography*. Comm ACM, 1975. 18:281-285

[20] Hubelbank M., Feldman C.L. *A 60x Computer-Based Holter Tape Processing System*. Med Inst, 1978. 12:324-326

[21] Ishijima M., Shin S.B., Hostetter G.H., Sklansky J. *Scan-Along Polygonal Approximation for Data Compression of Electrocardiograms*. IEEE Trans Biomed Eng, 1983. 30:723-729

[22] Jalaleddine S.M.S., Hutchens C.G., Strattan R.D., Cobberly W.A. *ECG Data Compression Techniques - A Unified Approach*. IEEE Trans Biomed Eng, 1990. 37:329-343

[23] Jiapu P., Tompkins W.J. *A Real Time QRS Detection Algorithm*. IEEE Trans Biomed Eng, 1985. 32:230-236

[24] Jiménez J.R., Martínez A., Pedraza J., Téllez V., Yáñez O. *Unidad Portátil para Electrocardiografía Ambulatoria Digital*. Rev Mex Ing Biom, 1992. 13:177-183

[25] Knoebel S.B., Lovelace D.E. *A Two Dimensional Clustering Technique for Identification of Multiform Ventricular Complexes*. Med Inst, 1978. 12:332-333

[26] Knuth D.E., *Semantics of Context-Free Languages*. Math Syst Theory, 1968. 2:127-146

[27] Lee H.S., Cheng Q.L., Thakor N.V. *ECG Waveform Analysis by Significant Point Extraction I - Data Reduction*. Comp Biom Res, 1987. 20:410-427

[28] Lee H.S., Cheng Q.L., Thakor N.V. *ECG Waveform Analysis by Significant Point Extraction II - Pattern Matching*. Comp Biom Res, 1987. 20:428-442

[29] Lin K.P., Chang W.H. *QRS Feature Extraction using Linear Prediction*. IEEE Trans Biomed Eng, 1989. 36:1050-1055

[30] Marques J.P., Abreu L.C. *A New ECG Classifier based on Linear Prediction Techniques*. Comp Biom Res, 1986. 19:213-233

[31] Mead C.N., Moore S.M., Clark K.W., Spenner B.F., Thomas L.J. *A Detection Algorithm for Multiform*

Premature Ventricular Contractions. Med Inst, 1978. 12:337-339

[32] Pahlm O., Sornmo L. *Software QRS Detection in Ambulatory Monitoring - A Review*. Med Biol Eng Comput, 1984. 22:289-297

[33] Papakonstantinou G., Gritzali, F. *Syntactic Filtering of ECG Waveforms*. Comp Biomed Res, 1981. 14:158-167

[34] Papakonstantinou G., Skordalakis E., Gritzali F. *An Attribute Grammar for QRS Detection*. Pattern Recognition, 1986. 19:297-303

[35] Partee B.H., Meulen A., Wall R. "Mathematical Methods in Linguistics". Kluwer Academic, 1987.

[36] Pavlidis T., Horowitz S.L. *Segmentation of Plane Curves*. IEEE Trans Comput, 1974. 23:860-870

[37] Pietka E. *Feature Extraction in Computerized Approach to the ECG Analysis*. Pattern Recognition, 1991. 24:139-146

[38] Skordalakis E. *Recognition of Noisy Peaks in ECG Waveforms*. Comp Biom Res, 1984. 17:208-221

[39] Thakor N.V., Zhu Y.S. *Applications of Adaptive Filtering to ECG Analysis: Noise Cancellation and Arrhythmia Detection*. IEEE Trans Biomed Eng, 1991. 38:785-793

[40] Thakor N.V. *From Holter Monitors to Automatic Defibrillators: Developements in Ambulatory Arrhythmia Monitoring*. IEEE Trans Biomed Eng, 1984. 31:770-777

[41] Trahanias P., Skordalakis E. *Bottom-Up Approach to the ECG Pattern Recognition Problem*. Med Biol Eng Comput, 1989. 27:221-227

[42] Udupa J.K., Murthy I.S.N. *Syntactic Approach to ECG Rhythm Analysis*. IEEE Trans Biomed Eng, 1980. 27:370-375

[43] Yáñez O., Jiménez J.R., Martínez A., Pedraza J., Téllez V. *Detección de Complejos QRS Normales en Registros de Electrocardiografía Ambulatoria de 24 Horas*. Rev Mex Ing Biom, 1990. 11:63-74

[44] Yáñez O., Jiménez J.R., Martínez A., Pedraza J., Téllez V. *Estudio sobre dos Algoritmos de Compresión de Datos para Electrocardiografía Ambulatoria Digital*. Rev Mex Ing Biom, 1992. 13:131-142