Universidade Federal de Santa Catarina
Centro Tecnológico
Departamento de Informática e Estatística
INE 5328 Projeto em Ciência da Computação II

Reconhecimento de Padrões Sobre Sinais de ECG utilizando Técnicas Sintáticas

Aluno:

Ângelo dos Santos Melo

Orientador:
Prof. Dr. Rer. Nat. Aldo von Wangenheim

Co-orientador:
Prof. Dr. Olinto Varela Furtado

Membro da banca:
M.Sc. Gilson Anselmo de Araújo

FLORIANÓPOLIS
Fevereiro de 2003

SUMÁRIO

Resumo

Sinais biomédicos, como o ciclo de batimento cardíaco, apresentam comportamentos que podem ser representados por uma linguagem regular. Este trabalho acadêmico se propõe estudar técnicas de codificação de sinais de Eletrocardiograma em uma linguagem regular e sua classificação utilizando um conjunto de autômatos finitos, coletando automaticamente informações sobre a morfologia e o comportamento do sinal analisado.

Abstract

Biomedical signals like the cardiac beat cycle presents behavior that can be encoded into a regular language. In this academic work is studed the tecnics of electrocardiograms encoded into regular languages and its syntatic pattern recognition using a set of finite automata, geting information about the morphology and behavior of the analysed signal.

Palavras-chave: Eletrocardiograma, Reconhecimento de padrões, análise sintática.

1. INTRODUÇÃO E JUSTIFICATIVAS

A implementação de um sistema especialista para o auxílio na detecção de anormalidades e eventuais doenças no exame de pacientes cardíacos é de grande valia para situações em que nem sempre se dispõe de atendimento médico imediato, principalmente em regiões do interior do país onde os recursos e profissionais disponíveis para tal tarefa são quase sempre escassos. De outro ponto de vista um sistema que auxilie o profissional na sua tarefa de análise de exames de eletrocardiograma também pode acarretar um acréscimo na produtividade quando diminui o tempo gasto em tarefas que são costumeiramente repetitivas e que podem ser automatizadas.

1.1 O Projeto Cyclops

O Projeto Cyclops é uma parceria bi-nacional entre a Universidade Federal de Santa Catarina e a Universidade de Kaiserlautern na Alemanha, liderado pelos professores Dr.rer.nat Aldo von Wangenheim e Dr. Michael M. Ritcher cujo objetivo é desenvolver sistemas para a análise de imagens médicas que possam ser de utilidade prática clínica e auxilio ao diagnóstico médico. As aplicações desenvolvidas utilizam técnicas de Inteligência Artificial e Visão Computacional e englobam sistemas para Tomografia Computadorizada, Ressonância Magnética, Ultra-som etc. O projeto é formado por um grupo de pesquisadores e conta com a cooperação de parceiros médicos e industrias.

1.2 Objetivos

O objetivo deste trabalho é a implementação de um sistema que aplique técnicas de reconhecimento de padrões para classificação de sinais biomédicos.

1.3 Objetivos Específicos

Os objetivos específicos deste trabalho são:

Obj. a) A implementação de um browser para visualização de sinais de eletrocardiograma e seus atributos mais significativos ;

Obj. b) acrescentar à aplicação a compatibilidade com o padrão Dicom 3.0 provendo a facilidade de integração com sistemas de informação hospitalar;

Obj. c) a implementação do modelo de reconhecimento sintático de padrões proposto por Antti Koski em [2] no processo da análise do sinal.

1.4 Organização do Trabalho

Este trabalho apresentará um estudo sobre os sinais a serem classificados e a descrição do sistema de reconhecimento. Assim, no capítulo 2 será apresentado um estudo sobre os sinais de eletrocardiograma afim de que o leitor possa ficar melhor ambientado sobre o problema proposto. No capítulo 3 é apresentada uma breve descrição sobre o padrão ***DICOM*** descrevendo suas funcionalidades e facilidades. O capítulo 4 traz uma breve introdução sobre reconhecimento de padrões, suas aplicações e as principais técnicas utilizadas na análise de eletrocardiogramas. O capítulo 5 descreve o sistema de autômatos com atributos utilizado para a classificação de eletrocardiogramas. O capítulo 6 aborda a descrição da implementação, dificuldades encontradas e testes. Por fim o capítulo 7 apresenta as conclusões e trabalhos futuros.

2. O ELETROCARDIOGRAMA E SUAS CARACTERÍSTICAS

O Eletrocardiograma (ECG) é um dos exames mais usados em cardiologia. Surgiu na mesma época em que se descobriu o funcionamento eletromecânico do músculo cardíaco cujas células produzem uma corrente de energia que pode ser medida através da utilização de eletrodos sobre a superfície da pele.
Através do eletrocardiograma pode-se medir o fluxo da corrente elétrica das células do coração fornecendo assim um resultado gráfico de suas atividades. Se um coração for acometido por uma anormalia como um infarto, este deverá apresentar regiões onde as células musculares encontram-se inativas o que afetará diretamente o comportamento do fluxo elétrico e decorrentemente modificará o traçado do ECG. Assim, o ECG é um registro gráfico do funcionamento do aparelho cardíaco e fornece informações sobre freqüência e ritmo cardíacos, aumentos nas câmaras cardíacas, anormalidades na posição anatômica do coração, processos obstrutivos nas artérias coronárias e até mesmo intoxicações. A captação e análise dos sinais de Eletrocardiograma são usadas desde o século XIX. A técnica foi desenvolvida por Augustus Waller em 1887.

O ciclo cardíaco - Um ciclo de batimento cardíaco é composto por três fazes: Diástole, Sístole auricular e Sístole ventricular. Na Diástole o sangue enche as aurículas e cerca de 80% dos ventrículos. Estes enchem-se porque se dilatam e portanto a sua pressão no interior da cavidade fica inferior à das aurículas. Na Sístole Auricular as aurículas contraem-se pela ativação do nódulo SA e empurram o resto do sangue que contêm para os ventrículos. A Sístole Ventricular ocorre em uma fração de segundos após a sístole auricular e nesta fase ocorre a contração dos ventrículos devido ao impulso elétrico do nódulo AV. Assim que os ventrículos começam a contraírem-se, a pressão no interior destes aumenta e assim que excede a pressão nas aurículas, as válvulas entre os aurículos e os ventrículos fecham-se completamente.
Esta pressão faz também com que as válvulas à saída dos ventrículos se abram e assim o sangue corre para fora do coração para a artéria pulmonar ou para a aorta.

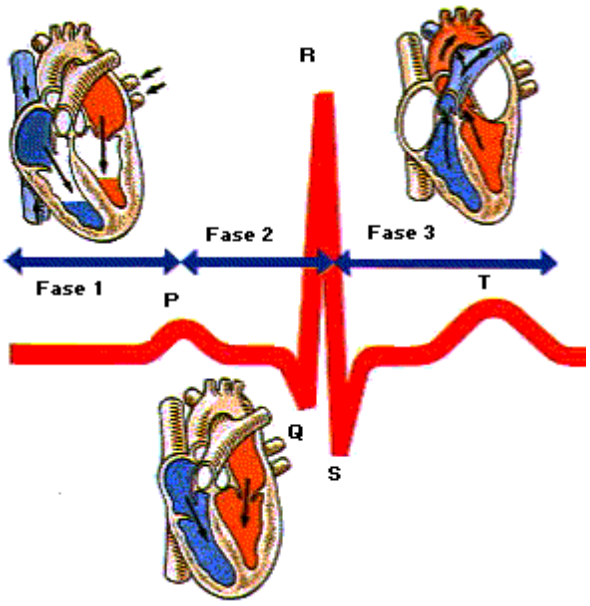


Figura 2.1 – As três fases do ciclo cardíaco e o sinal de ECG correspondente

O traçado do sinal - O formato mais genérico possui um conjunto de cinco ondas Q, P, R, S , T e eventualmente uma onda U. Outras características importantes no sinal são os intervalos e segmentos entre ondas como o intervalo RR que define o ciclo de um batimento cardíaco, e os seguimentos PQ, ST e TP.

A onda P representa o impulso elétrico da contração das aurículas. O complexo QRS representa o impulso da contração dos ventrículos e a onda T corresponde à recuperação elétrica dos ventrículos quando estes voltam para repouso. A análise do movimento e formato das ondas fornecem informação essencial do estado e da saúde do coração. A análise do sinal compreende a monitorização do ritmo cardíaco e a medição do intervalo entre batidas - o intervalo R-R. Em relação ao ritmo cardíaco, faz-se a média do número de ciclos cardíacos do coração durante um período de tempo.

Coleta dos sinais - Os sinais podem ser obtidos através das aplicações dos eletrodos em diferentes posições ou derivações.
Existem três tipos de derivações. Na Derivação I coloca-se um eletrodo no pulso direito e outro no pulso esquerdo de maneira que a polarização deste último eletrodo é positiva em relação ao eletrodo no pulso direito. Isto faz com que a onda R vá para cima no traçado ECG. Na derivação II, coloca-se um eletrodo no pulso direito e outro no tornozelo esquerdo, de maneira que a polarização no tornozelo é positiva em relação ao eletrodo no pulso. Este tipo de derivação produz uma onda R bastante alta .

A figura 2.2 demonstra alguns componentes e seguimentos de um sinal de ECG genérico.

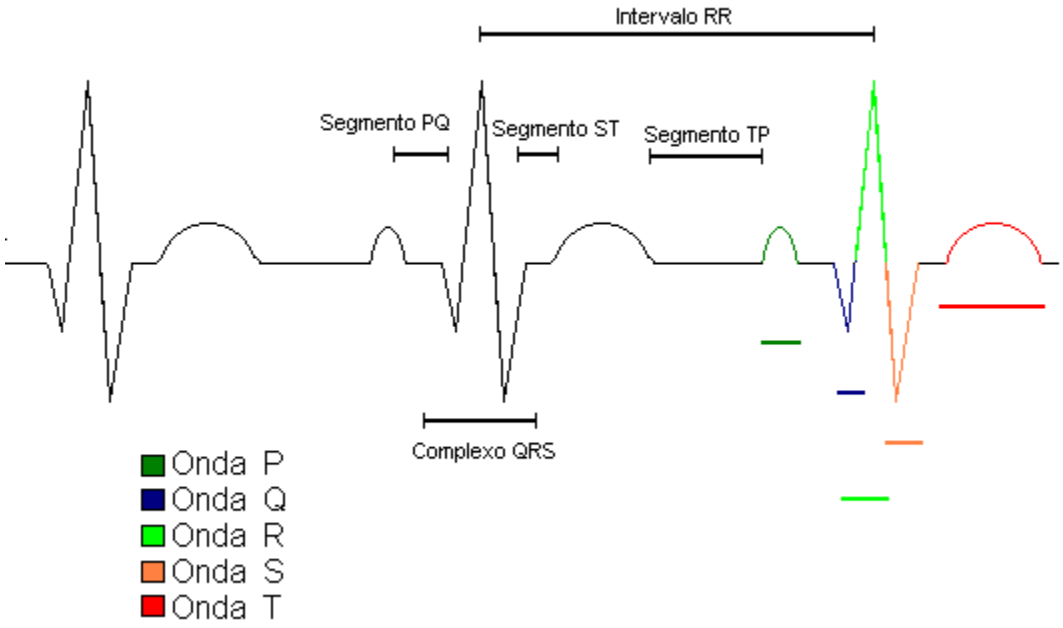


Figura 2.2 – Um modelo de ECG apresentando três ciclos e principais componentes.

2.1 Amostras de ECG

A seguir são apresentadas algumas amostras de ECG e seus respectivos diagnósticos conforme o padrão do traçado do sinal.

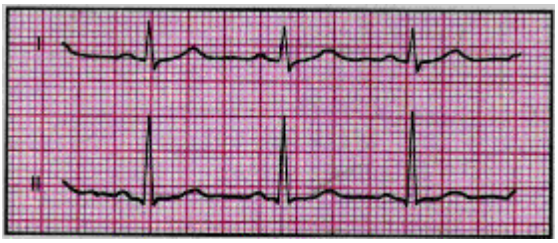


Figura 2.1.1 – Ritmo Sinusal normal

Figura 2.1.2 – Bloqueio Atrioventrícular de primeiro grau

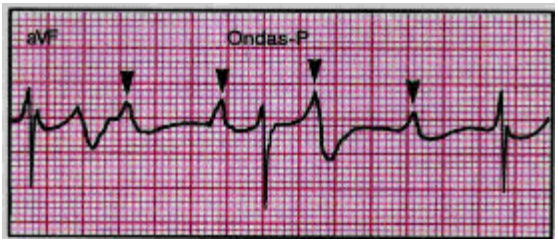


Figura 2.1.3 – Bloqueio Atrioventrícular completo

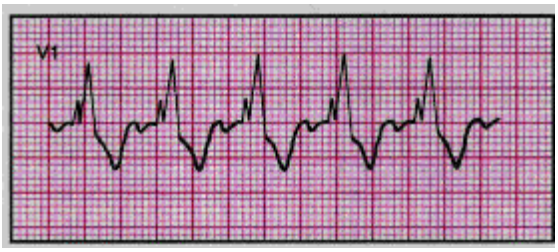


Figura 2.1.4– Bloqueio de ramo direito

Figura 2.1.5 – Infarto do Miocárdio inferior

Figura 2.1.6 – Infarto do Miocárdio sem onda Q

Figura 2.1.7 – Fibrilação Atrial

Como visto nas amostras acima, o sinal de ECG pode variar muito, apesar de possuir seus componentes característicos, as ondas P, Q, R, S e T. Dependendo da posição dos eletrodos ou de algum tipo de anormalia no sinal alguns subpadrões podem eventualmente ficar ausentes nas amostras.

O componente mais comum nos gráficos é o complexo QRS estando presente em praticamente todas as amostras, embora casualmente apresentando uma morfologia modificada.

2.2 Componentes de um sinal de ECG

2.2.1 Onda P

A onda P deve ser analisada quanto às seguintes características:
-Eixo (orientação): no plano frontal o eixo de P fica entre 0º e 90º (onda P positiva em DI, DII e aVF e negativa em aVR), considerado o vetor normal dirigido para baixo e para a esquerda. No plano horizontal, o vetor se dirige para frente (onda P positiva em V1). Em V1, a onda P pode ser difásica tipo plus-minus. Quando isso ocorre a fase positiva deve ser maior do que a negativa.
-Amplitude: a maior amplitude não deve exceder 2,5 mm (0,25 mV).
-Morfologia: arredondada e monofásica, podendo ser difásica em V1.
-Duração: duração máxima é de 0,10s.

2.2.2 Intervalo PR

É medido do início da onda P até o início do QRS. Varia de 0,12s a 0,20s. Representa o tempo que o impulso gerado pelo NSA levou para atingir as fibras de Purkinje.

2.2.3 O Complexo QRS

O complexo QRS deve ser analisado quanto às seguintes características:
-Eixo (orientação): a faixa de variação do eixo do QRS no plano frontal é de -30º a +120º. No plano horizontal, o vetor médio do QRS é orientado para trás.
-Amplitude*:* diz-se que existe baixa voltagem quando não se registra qualquer deflexão maior do que 5mm em derivação bipolar ou se a maior deflexão no plano horizontal não ultrapassa 8mm. Alta voltagem é definida quando se registra ondas R ou S > 20mm nas derivações frontais ou, no plano horizontal, ondas S (V1/V2) ou ondas R (V5/V6) > 30 mm.
-Morfologia*:* varia de acordo com a derivação e a posição elétrica do coração. Onda Q. É a primeira deflexão negativa do QRS e representa a ativação septal. Onda Q patológica é definida quando exceder 25% do tamanho de R e duração > 0,04s. Em algumas derivações, estes limites podem ser ultrapassados (aVR, aVL e D3). A presença de onda Q em V1, V2 e V3 deve ser sempre considerada anormal. A ausência de onda Q em V5 e V6 também é anormal.

2.2.4 Onda R

É a primeira deflexão positiva do QRS e representa fundamentalmente a ativação das paredes livres. Normalmente deve progredir de amplitude de V1 para V6.

2.2.5 Onda S

É a segunda deflexão negativa do complexo QRS e representa a ativação das porções basais dos ventrículos. Normalmente deve diminuir de amplitude de V1 para V6.
-Duração*:* o complexo QRS deve ter duração máxima de 0,12s. Deflexão intrinsecóide é o tempo de ativação ventricular. Medido do início do QRS até o vértice da onda R, deve ser no máximo de 0,045s. O aumento da deflexão intrinsecóide pode ocorrer por: hipertrofia ventricular, bloqueio de ramo, bloqueio divisional ou infarto agudo do miocárdio.

2.2.6 Segmento ST

Começa no ponto J (término do QRS) e termina na porção ascendente da onda T. Normalmente a primeira porção do segmento ST é isoelétrica. Desníveis do segmento ST podem ocorrer por múltiplas causas, sejam elas primárias (corrente de lesão do IAM) ou secundárias (hipertrofias, bloqueios de ramo, etc...).

2.2.7 Onda T

Sua orientação segue o vetor médio do QRS. Tem morfologia tipicamente assimétrica, com a porção inicial mais lenta. Não deve exceder 5mm nas derivações frontais ou 10 mm nas precordiais. Sua polaridade pode ser muito variável, sendo obrigatoriamente positiva em V5 e V6 e obrigatoriamente negativa em aVR.

2.2.8 Intervalo QT

É medido do início do QRS até o final da onda T e representa o tempo de ativação e recuperação do miocárdio ventricular. O QT varia com a idade, sexo e muito com a freqüência cardíaca;
Portanto, deve ser corrigido através da fórmula de Bazzet:
(O limite superior para homens fica em torno de 0,425s e para mulheres em torno de 0,440s).

3. O PADRÃO DICOM 3.0 E AMOSTRAS DE ECG

DICOM – **D*igital Imaging Communication in Medicine*** é um padrão para digitalização de amostras e imagens médicas. Foi desenvolvido por um consórcio formado pelo Colégio Americano de Radiologia (ACR) e a Associação Nacional de Fabricantes de Equipamentos Elétricos (NEMA) com o objetivo de permitir a compatibilidade na comunicação de equipamentos de imagens médicas digitais (Raio-x, Tomógrafo Computadorizado etc.). Sua primeira versão data de 1985. Uma segunda versão, o DICOM 2.0, foi desenvolvida em 1988 para corrigir alguns problemas da primeira versão quando o padrão começou a se difundir. A terceira versão, o DICOM 3.0, foi completada em 1992 e apresenta várias melhorias em relação as versões anteriores e sua maior funcionalidade é a transferência de imagens médicas sem problemas de compatibilidades de sistemas de comunicação, isto porque a arquitetura do protocolo DICOM foi feito para ser independente de plataforma. Outro objetivo desta versão é o desenvolvimento e expansão dos PACS do inglês Picture Archieving and Communications System e interfaceamento com sistemas de informação Médicos [8].

O padrão DICOM 3.0 está muito difundido e tem sido utilizado tanto pela industria de equipamentos quanto pelos desenvolvedores de software.

O comitê ACR-NEMA (American College of Radiology and National Electrical Manufactures Association) responsável pela elaboração do padrão DICOM está dividido em diversos grupos de trabalho dentre os quais está o grupo de trabalho 1 - Cardiac and Vascular Information que é responsável pelo desenvolvimento de novas modalidades. No ano de 2000 foi introduzido ao padrão a modalidade Waveform [9] que fornece ao protocolo DICOM a capacidade de contemplar exames cujo resultado não se apresenta como uma imagem de pixels, mas sim como ondas, como exemplo o ECG, HD e Áudio(AU) [8].

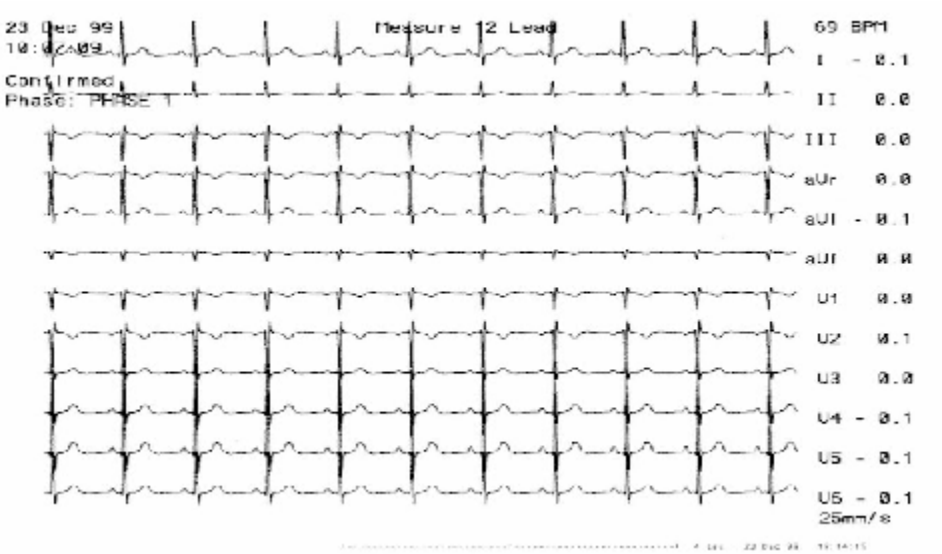


Figura 3.0 – Exame de uma ECG de doze canais seguindo a codificação Dicom.

4. RECONHECIMENTO DE PADRÕES E ANÁLISE DE SINAIS DE ECG

Reconhecimento de padrões - reconhecimento de padrões é uma técnica computacional que tem sido alvo de grandes pesquisas ultimamente na comunidade de cientistas da computação mundial.

O objetivo principal é a classificação de uma instância qualquer utilizando-se ou não um conjunto de dados predisposto como comparação. Para isso emprega diversas técnicas dependendo das características dos dados analisados, entre elas aprendizado de máquina, técnicas de IA conexionista e simbólica, análises estatísticas e medidas de distância como *Hamming* e *Nearest Neighbour*.

A técnica de reconhecimento de padrões pode ser aplicada em reconhecimento de imagens, sons, características comportamentais dentro de um conjunto de dados como um texto, grupo de pessoas etc. São inúmeras as suas aplicações e atualmente tem sido empregada em áreas de negócio, através da análise comportamental de dados comerciais como características de um novo cliente ou a movimentação da bolsa de valores, e principalmente em aplicações científicas, onde são empregadas diversas técnicas de análise numérica e estatística de dados como, por exemplo, a classificação de elementos químicos que compõe a constituição de uma estrela através da análise do espectro de freqüência de sua luz e a detecção de anormalidades na análise de uma imagem de tomografia computadorizada.

Sem dúvida, uma de suas aplicações de mais utilidade é a análise de dados biomédicos, pois auxiliam o profissional em sua tarefa de atingir um diagnóstico para algum sintoma. Atualmente emprega-se técnicas de reconhecimento de padrões na análise de exames de tomografia computadorizada, análise de tecidos em mamografia e análise de sinais biomédicos como eletrocardiograma, eletroencefalograma, eletromiografia etc.

Análise e diagnóstico de Eletrocardiogramas - A análise de eletrocardiogramas é uma das tarefas mais complexas de um cardiologista, isto porque as amostras podem apresentar inúmeras particularidades que, na maioria das vezes podem se apresentar de maneira muito sutil. Por isto existe a necessidade de tal tarefa ser desempenhada por um especialista no assunto. Alguns exames chegam até a ser alvo de discussões em fóruns de preparados para este fim. Talvez a implementação de um sistema para o fornecimento automático de diagnósticos seja impossível de ser implementado, mas a extração automática de parâmetros de interesse para o profissional pode ser alcançada e é justamente em torno deste objetivo que as pesquisas de análise de sinais biomédicos tem voltado seu foco.

Reconhecimento de padrões e sinais biomédicos - Atualmente existem várias pesquisas sobre a classificação de sinais biomédicos especialmente sobre amostras de ECGs, apesar da complexidade de tal tarefa. Existem vários artigos sobre o assunto onde são aplicadas as mais diversas técnicas de reconhecimento de padrões desenvolvidas.

O objetivo principal da análise automatizada de ECGs está em reconhecer com precisão os principais subpadrões (ondas P e T e complexo QRS) afim de extrair os seus parâmetros de duração e amplitude.

Dentre as técnicas desenvolvidas nas últimas décadas algumas são baseadas em métodos numéricos como a transformada de Fourier, e técnicas de IA como algoritmos genéticos, lógica Fuzzy e redes neurais.

Sinais são quantidades físicas detectáveis que transmitem informações e podem ser representados de duas formas básicas: Representação no domínio do tempo, onde a amplitude do sinal é representada como uma função do tempo e representação no domínio da freqüência, onde a função que representa o sinal mostra a amplitude de cada freqüência que o compõe.

Um sinal de eletrocardiograma consiste em amostras dos potenciais elétricos do coração sendo uma representação no domínio do tempo.

Os métodos numéricos de reconhecimento consistem na análise do sinal utilizando algum tipo de transformação ou aproximação e são mais indicados quando se utiliza sinais representados no domínio da freqüência, onde se deseja capturar informações contidas no espectro do sinal.

4.1 PRINCIPAIS PARADÍGMAS UTILIZADOS NA ANÁLISE DE SINAIS DE ECG

4.1.1 Análise utilizando a Transformada de Fourier

Desde 1807, a análise de sinais utilizando a transformada de Fourier dominou o campo da representação de sinais no domínio da freqüência. Porém este tipo de análise não possui poder de representação suficiente quando os sinais analisados são não periódicos. Isto se dá principalmente porque a Transformada de Fourier não fornece uma boa localização das freqüências que compõem o sinal no domínio do tempo [6]. Em [8] é apresentada uma aplicação onde a transformada de Fourier é utilizada para realizar a compressão do sinal afim de reduzir os requisitos de armazenamento e transmissão de eletrocardiogramas digitalizados.

4.1.2 Análise utilizando a Transformada de WAVELET

A teoria de Wavelets foi desenvolvida justamente para aumentar o poder de representatividade deficiente da análise de Fourier. A transformada de Wavelets tem trazido resultados satisfatórios em áreas como compressão de dados, detecção de características em imagens e remoção de ruídos de sinais físicos e biológicos [6].
Em [3] é apresentado um modelo de aplicação para reconhecimento de padrões em sinais de ECG para caracterização de arritmias cardíacas e outras irregularidades, onde vários processos de reconhecimento foram utilizados dentre eles a transformada de Wavelets. Nota-se que a transformada de Wavelets é muito utilizada no processamento de sinais para resolução de parte do problema na analise de ECG, mas a transformada Wavelets é uma teoria relativamente nova, e tem se revelado uma ferramenta poderosa no processamento e análise de sinais para inúmeras aplicações.

4.1.3 Redes Neurais Artificiais

Existe uma grande quantidade de pesquisas utilizando redes neurais no processamento de sinais de ECG digitais afim de aproveitar suas vantagens de possuir a capacidade de aprendizado e treinamento.

Em [4] é apresentado uma implementação onde redes neurais foram empregadas na classificação e diagnóstico de infarto agudo do miocárdio, utilizando o ECG de 12 derivações, onde características dos ECGs foram extraídas usando as técnicas de análise do componente principal, que permite um número pequeno de indicadores eficazes.

Em [5] uma rede neural artificial do tipo *feedforward/backpropagation* é capaz de efetuar a classificação dos segmentos do ECG e é usada na representação de características do segmento ST para detecção automática de isquemias miocárdicas.

Em [4] a rede neural é usada na classificação de arritmias e reconhecimento de doenças crônicas do miocárdio.

4.1.4 Análise utilizando Lógica Fuzzy

A arquitetura de rede neurofuzzy é aplicada no reconhecimento de padrões específicos no ECG. Em [1] foi desenvolvido um sistema de redes neurofuzzy para detecção da onda P em amostras de ECG onde o algoritmo de treinamento da rede foi feito usando algoritmos genéticos, o que fez com que a rede fosse rapidamente treinada, porém a taxa de reconhecimento de ondas P diminui com o aumento de ruído no sinal. No processo de localização das ondas P as amostras coletadas dos sinais são seqüencialmente aplicadas à entrada da rede neurofuzzy, onde a primeira entrada recebe a amostra mais recente do sinal e a última entrada recebe a amostra mais antiga. Neste trabalho um total de três testes completos e independentes foram realizados usando arquivos de sinais previamente adquiridos. Os resultados foram satisfatórios pois o sistema encontrou grande quantidade de ondas P. Porém houve várias ondas P que não foram detectadas tendo sido confundidas com ruído devido a atenuação do sinal. Também houve casos em que a onda T foi reconhecida como uma onda P o que mostra a falta de um relacionamento sintático entre os subpadrões classificados.

4.1.5 Análise sintática do sinal

O reconhecimento sintático de padrões é baseado na composição estrutural de padrões a partir dos padrões mais simples do sinal. As partes mais elementares dos padrões são chamadas de primitivas e então existem relações para construir padrões mais complexos a partir de padrões primitivos mais simples[7].

As técnicas de análise sintática dos dados representados pelas amostras apresentam a vantagem de requererem menos processamento por não trabalharem com o sinal no domínio da freqüência. Além disso, a natureza dos sinais biomédicos quase sempre define uma estrutura sintática regular[2]. Como a natureza destes sinais são geralmente muito complexas e a interpretação da sua estrutura depende do contexto onde elas ocorrem, a análise do sinal pode ser reconhecida pela aplicação de um parser que reconheça uma linguagem definida pelo sinal. Um problema é a construção de gramáticas e do analisador sintático para o sinal pois tem-se que usar na maioria das vezes linguagens muito complexas. Para resolver isto em [2] foram adicionadas heurísticas na analise sintática através da utilização de atributos para primitivas e estados. Neste sistema foi aplicado um conjunto de autômatos com atributos para desempenhar a tarefa de reconhecimento sintático de sinais de ECG. Os atributos fornecem informação adicional pois podem expressar características numéricas não estruturais inerentes do sinal e consequentemente diminuir a complexidade da formalização de uma gramática para representar o sinal.

4.2 Conclusões

Os resultados alcançados com a utilização dos métodos sintáticos apresentaram boa performance e índices satisfatórios. Os experimentos realizados em [2] apresentaram 99% de sucesso na detecção de complexos QRS, 91% na detecção de ondas T e 77% na detecção de ondas P. Estes dois últimos subpadrões tiveram uma taxa de reconhecimento um pouco menor devido a atenuação do sinal, principalmente no caso de ondas P que são ondas de pequena amplitude e quase sempre aparecem muito tênues nas amostras. Outro fator é o ruído que pode distorcer o sinal fazendo com que alguns subpadrões fiquem corrompidos.

Os trabalhos utilizando redes neurais apresentaram bons resultados, mas são limitados a conjuntos de sinais específicos como arritmias.

Assim, foi escolhido o paradigma de análise sintática para a implementação de um sistema de classificação de sinais de ECG baseado no sistema proposto por Antti Koski em [2].

5. SISTEMA DE AUTÔMATOS FINITOS

Como foi discutido no capítulo 4, as técnicas sintáticas de reconhecimento de padrões são muito indicadas quando o sinal a ser analisado é de natureza biomédica pois estes quase sempre definem uma estrutura sintática regular. Neste capítulo será apresentada a descrição do sistema de reconhecimento proposto por Antti Koski para reconhecimento sintático de padrões, visto ser a técnica abordada que apresentou maior número de resultados satisfatórios em testes sobre eletrocardiogramas.

Análise do sinal - A análise do sinal é feita em duas fases: Na primeira fase o sistema faz a aquisição do sinal seguindo um passo de amostragem arbitrário e faz a computação das primitivas e seus atributos de tempo e amplitude sobre cada unidade de amostra. Na segunda fase, o sistema faz a análise sintática das primitivas obtidas na primeira fase, fazendo a classificação do sinal e o cálculo final das propriedades dos subpadrões.

5.1 Computação das primitivas

O arquivo Dicom fornece uma lista de valores de amplitudes do sinal a cada passo de amostragem que segue uma freqüência de 400 Hz. Após a leitura destes valores é formado um par ordenado onde a coordenada x é um valor de tempo incremental e a coordenada y é o valor da amplitude, definindo assim um ponto sobre o traçado do sinal.

A computação das primitivas foi feita através da análise da inclinação dos segmentos de reta definidos a cada par de pontos consecutivos do sinal e as diferenças dos níveis de tensão(unidade de amplitude) e o intervalo de tempo definido pelos dois pontos(unidade de duração).

Assim, cada primitiva é um registro que contém os seguintes atributos:

- ***Label*** : um elemento do alfabeto da linguagem que é função da inclinação da reta formada por dois pontos consecutivos do sinal;
- ***Amp***: a amplitude ou nível de tensão da amostra e
- ***Dur*** : intervalo de tempo da amostra.

Seguindo o modelo proposto por Antti Koski, foi estabelecido um conjunto de cinco primitivas, cada uma representando o ângulo de inclinação dos segmentos como mostra a figura 5.1.1.

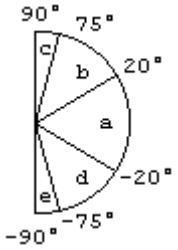


Figura 5.1.1 – Distribuição dos labels em função dos ângulos dos segmentos.

Cada primitiva recebe o label conforme o seu ângulo de inclinação estiver presente em um dos cinco intervalos estabelecidos.

O cálculo dos atributos de amplitude e duração são facilmente obtidos através da diferença dos valores das coordenadas de dois pontos consecutivos. Assim, dado um seguimento (x1,y1) , (x2, y2) seu valor de amplitude é dada por y2 – y1 e o seu valor de duração por x2 – x1.

5.2 O sistema de autômatos com atributos desenvolvido por Antti Koski

Um autômato com atributos é um autômato finito onde foram adicionados variáveis escalares chamados atributos aos estados. Este possui um estado inicial e uma função de transição parcial dos estados e símbolos de entrada que mapeia estados como um autômato convencional. Adicionalmente existem juntamente com os estados os atributos, e com cada função de transição existe uma função de avaliação que calcula os atributos do próximo estado da transição. Esta função de transição usa os atributos do estado anterior e os atributos de uma primitiva de entrada como argumentos. Além disso, um autômato com atributos possui um predicado booleano que utiliza o estado atual e os seus atributos como argumento. Este predicado indica a aceitação de uma string de entrada e recebe o valor verdade se e somente se o autômato aceitar sua entrada. Como a função de transição é parcial, o autômato pode parar embora a string de entrada não tenha sido totalmente analisada. É definido então que o autômato aceitou um determinado prefixo de uma string de entrada se o predicado booleano for igual a verdadeiro, diferentemente de autômatos convencionais em que este predicado de aceitação é uma função característica do conjunto de estados finais. Para cada autômato com atributos é preciso especificar os atributos iniciais do estado inicial antes de ativá-lo.

Atributos - Atributos são variáveis que contém informações calculadas sobre a estrutura sintática durante o procedimento de parsing. Esta informação consiste em valores numéricos de amplitude e duração das primitivas de entrada, que são usados para decidir se o autômato aceita a sentença de entrada ou não.

Conjunto de Autômatos - Consequentemente, foi organizado um conjunto de autômatos em que cada autômato possui uma tarefa particular no processo de reconhecimento. Qualquer autômato pode chamar outro autômato no conjunto ou a si mesmo recursivamente para desempenhar alguma tarefa especifica de reconhecimento. Assim um autômato pode transitar de um estado para outro, baseado no estado atual e no símbolo na entrada, ou chamar um sub-autômato em qualquer estado, no processo de reconhecimento.

Quando um autômato U pára e chama um autômato V, este usa os atributos do autômato U para inicializar os atributos do seu estado inicial e começa a processar a sentença de primitivas de entrada a partir da posição L1 onde U chamou V. O autômato V tenta reconhecer o subpadrão para qual foi designado. Se um autômato V reconhece um subpadrão ele pára e o autômato U recebe os atributos de V e continua o processo de reconhecimento a partir da posição L2 onde V parou. Caso V falhe, o autômato U continua o processo usando outras alternativas a partir da posição L1. No conjunto de autômatos existe um autômato inicial que começa todo o processo. O trabalho do autômato V pode terminar por dois motivos:

- Primeiro, a sentença de entrada termina, assim todo o processo termina também.
- Segundo, pelo motivo de a função ser parcial, pode não haver mais transições ou pode haver uma chamada para um autômato não considerado, em um certo estado durante o processo. Isto significa que a estrutura sintática, para a qual V foi designado, não continua mais na string de primitivas.

Definição formal do sistema de autômatos - O sistema de autômatos com atributos é definido como a seguinte tupla:

(**M, m0, Q, I,** Σ**,** Γ**, F, T,** μ**0**)

onde:

M é o conjunto não vazio de autômatos, e *m0* é o autômato inicial.

Q é o conjunto não finito de estados e *Qm* se refere aos estados de um autômato *m*. Todos os conjuntos *Qm* são disjuntos entre os autômatos.

I é o conjunto de estados iniciais dos autômatos (**I** $\subseteq$**Q**) e todos os autômatos possuem exatamente um estado inicial Im por autômato m .

Σ é o alfabeto de primitivas do sistema.

Γ é um conjunto finito de atributos dos estados e Γp se refere aos atributos do estado *p*. Toda primitiva também possui atributos e Γc se refere ao atributos da primitiva *c*.

F é um conjunto de regras de computação dos atributos. A relação de transição T consiste de dois componentes T1 e T2. O componente T1 é uma função parcial determinística de **M x Q x** Σ **x F** em **Q** e que mapeia transições dentro de um autômato. A tupla (u, p, c, f, q) de T1 é a transição para um autômato U do estado p para o estado q com o símbolo c na entrada e os atributos do estado q são avaliados com f, usando os atributos do estado p e o símbolo de entrada c, ou seja, e $\Gamma q = f (\Gamma p, \Gamma c)$.

O componente T2 é uma relação de transição não determinística **M x Q x Σ x F x F x M x Q**, que descreve as ações do sistema entre os autômatos. A tupla (u, p, c, f, g, v, q) de T2 é uma chamada do autômato v, a partir do estado p do autômato u com o símbolo c na entrada. Os atributos iniciais do autômato v são calculados com f : ΓIv = ***f*** (Γp) Se o autômato V reconhecer seu subpadrão e o seu trabalho terminar no estado r, o sistema retorna para o estado q do autômato u e avalia os atributos de q com a função g : Γq = g (Γr). Assim a cada transição que um autômato faz, os atributos do estado atual são usados para computar os atributos do estado subseqüente. Quando o sistema retorna com sucesso do subautômato, os atributos computados pelo subautômato são usados para calcular os atributos do estado para qual o sistema retornou.

μ0 contém os valores iniciais dos atributos do estado inicial do autômato inicial no momento de inicialização do sistema.

O autômato em um nível mais alto pode ter várias alternativas de escolha dentro de alguma fase do parsing. As ações do sistema foram definidas formalmente conforme a seguinte tupla:

(u,p,w,S)

Onde:

u é o autômato atual a ser processado
p é o estado atual dentro do conjunto Qu,
w é a sentença de entrada ainda não analisada, e
S é a pilha de controle do sistema.

Uma configuração inicial é (m0, Im0, w, ε), onde w é a sentença completa de entrada e ε é a pilha vazia e Γ Im0 = μ0. O predicado de aceitação do autômato u será referenciado por Pu. Estes são os quatro casos para as mudanças de configuração do sistema: (|| = concatenação)

(i) (u,p,cw,S) $\rightarrow$ (v, Iv, cw,(u,p,cw,g,q)||S),
se (u,p,c,f,g,v,q) está em T2 e Γ Iv = f(Γ p).

(ii) (u,p,cw,S) $\rightarrow$ (u,q,w,S),
se (u,p,c,f,q) está em T1 e Γ q = f(Γ p, Γ c).

(iii) (u,p,cw,(v,q,z,g,r)||S) $\rightarrow$ (v, r, cw,S),
se Pu(p, Γ p) é verdadeiro e Γ r = g(Γ p) e não há transições ou o subautômato faz uma chamada não considerada no autômato u.

(iv) (u,p,cw,(v,q,z,g,r)||S) $\rightarrow$ (v, q, z, S),
se Pu(p, Γ p) for falso e não há transições ou o subautômato faz uma chamada não considerada no autômato u.

Enquanto o sistema estiver em certa configuração poderá haver várias alternativas para chamar um subautômato (i) pois T2 é não determinística. Primeiro o sistema tenta todas as alternativas de T2 seguindo uma ordem

predeterminada. Se todas falharem o sistema checa se existe alguma transição não determinística disponível em T1 (ii). No momento da chamada do subautômato (i) a informação sobre o parsing é guardada na pilha. Isto inclui o nome do subautômato atual, o estado, a sentença de entrada ainda não processada, as regras de avaliação dos atributos e o estado de retorno após um retorno bem sucedido. Se o parsing falhar (iv), o sistema retorna para o autômato chamador e continua seu trabalho a partir da próxima alternativa de chamada, se ainda houver alternativas. O sistema irá aceitar a primeira chamada de autômato bem sucedida (iii) e continua seu trabalho a partir do próximo estado r ignorando as alternativas restantes do estado anterior q. A ordem em que um subautômato é chamado é organizada apropriadamente dentro da aplicação, assim os subpadrões mais importantes são pesquisados antes dos menos importantes.

Quando toda a sentença tiver sido analisada, o sistema assume a configuração final (u, p, ε, S), onde ε é a string nula. Se o autômato u não for m0, então o registro mais baixo de S que é sempre o registro de m0 tem que ser buscado da pilha. Os valores dos atributos do estado atual de m0 contém informação sobre os cálculos da duração média e da amplitude dos subpadrões de toda sentença de entrada.

5.3 O sistema de reconhecimento de ECG

Foi modelado um sistema de dez autômatos para a tarefa de reconhecimento de ECG, em que cada autômato é especializado para um subpadrão especifico. Durante o processo cada autômato calcula a duração e amplitude do subpadrão sendo processado. Estes valores são facilmente calculados a partir das primitivas onde existem registros de duração e amplitude para cada primitiva individualmente. Após um subpadrão ter sido reconhecido com sucesso, sua amplitude e duração é repassada para um autômato chamador em um nível mais alto. O sistema também calcula a média corrente de amplitude e duração das primitivas do sinal. Estes dois valores médios são então usados na determinação dos critérios de reconhecimento de subpadrões diferentes.

O autômato inicial ECG coleta informações médias sobre parâmetros importantes do sinal de ECG. Sua primeira tarefa é encontrar o primeiro complexo QRS em um sinal e dar inicio a analise de batidas consecutivas. O autômato ECG calcula a amplitude média e a duração de ondas P, T, complexos QRS, seguimentos ST e TP e PT e intervalos RR. Ele usa o autômato BEAT que analisa cada ciclo cardíaco do sinal. Um ciclo é então definido entre um complexo QRS, excluindo este, até o próximo QRS incluindo este ultimo.

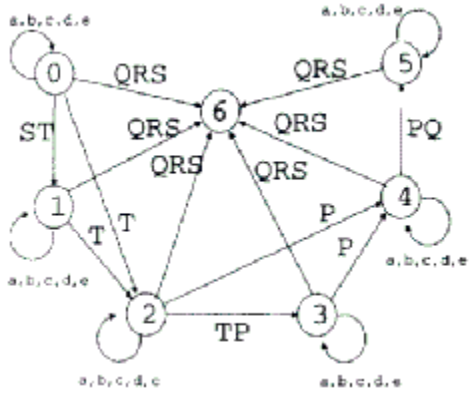


Figura 5.3.1 – Autômato BEAT encarregado de fazer a análise sintática de cada ciclo do batimento cardíaco

Cada estado do autômato BEAT corresponde a uma fase da analise de um ciclo cardíaco. A principio, no estado 0 a analise de um segmento ST é testado se for encontrada uma primitiva *a* . Se a análise do segmento ST for bem sucedida, o sistema retorna para o estado 1, se falhar, o sistema tenta analisar uma onda T se forem encontradas primitivas *b* ou *d*. Se for bem sucedido, o sistema retorna para o estado 2, se falhar o sistema tenta analisar um complexo QRS com as primitivas *c* ou *e*. Se este for reconhecido, o sistema volta para o estado 6 , a partir do qual não há mais transições e assim o sistema retorna com sucesso para o autômato BEAT. Se todas as alternativas falharem o sistema avança um símbolo na sentença de entrada, permanece no estado 0 e tenta novamente analisar as alternativas apresentadas.

No estado 1 o sistema tenta reconhecer primeiro uma onda T, depois um complexo QRS. No estado 2 o sistema tenta reconhecer primeiro um segmento

TP, depois uma onda P, e por fim um complexo QRS. e assim por diante conforme o diagrama na figura *5.3.1*.

Após um parsing bem sucedido de um subpadrão, os atributos produzidos pelo subautômato são copiados para o conjunto de atributos do autômato Beat. Os atributos descrevem as durações e amplitudes das ondas T e P, dos seguimentos ST, TP e PQ e dos complexos QRS.

Os autômatos ST, TP e PQ reconhecem seguimentos planos. Sua estrutura sintática é dada pela expressão regular *a*+ e eles calculam a duração e a amplitude do seguimento plano. Se um seguimento ST *a*n for analisado, a duração e a amplitude do seguimento é calculada diretamente a partir das primitivas *a* conforme a equação 8.

O reconhecimento de um seguimento plano é bem sucedido se a amplitude do seguimento estiver abaixo da amplitude média e das primitivas do sinal e a duração estiver acima da duração média das primitivas do sinal.

$$\text{Dur (ST)} = \sum_{i=1}^{n} \text{Dur}(a_i) \quad \text{e}$$

$$\text{Amp (ST)} = \sum_{i=1}^{n} \text{Amp}(a_i) \qquad (8)$$

O autômato TWAVE analisa ondas T e possui a estrutura sintática dada pela seguinte expressão regular: b+ a* (d|e)+ |d+ a*(b|c)+ que descreve a subida e decida das formas de onda. Se uma onda T for analisada com a seguinte estrutura *b*n1 *a*n2 *d*n3 a duração da onda conforme a equação 9:

$$\text{Dur (T)} = \sum_{i=1}^{n1} \text{Dur}(b_i) + \sum_{i=1}^{n2} \text{Dur}(a_i) + \sum_{i=1}^{n3} \text{Dur}(d_i) \qquad (9)$$

Para calcular a amplitude da onda no seguimento analisado, é calculado a altura do lado esquerdo da onda Ampl(T) e a altura do lado direito da onda Ampr(T), e a média destas alturas fornece a amplitude média da onda T:

$$\text{Ampl (T)} = \sum_{i=1}^{n2} |\text{Dur}(b_i)| \qquad (10)$$

$$\text{Ampr (T)} = \sum_{i=1}^{n3} |\text{Dur}(d_i)| \qquad (11)$$

$$Amp\ (T) = (Ampl\ (T) + Ampr\ (T))/ 2 \qquad (12)$$

A análise da onda T é bem sucedido se as seguintes condições forem satisfeitas:
Primeiro, a magnitude das alturas Ampl (T) e Ampr (T) dos lados esquerdo e direito estão perto um do outro (fig7), o que pode ser expresso de maneira simples com o uso dos atributos.
Segundo, a duração Dur (T) da onda T reconhecida deve ser uma determinada fração da média corrente do intervalo RR. A duração DURt (T) da área plana sobre o topo da onda (equação13) deve ser menor que a metade da duração total Dur (T) da onda T.

Também é obrigado que o ponto de inicio da onda T dentro do ciclo cardíaco atual esteja abaixo 0.2 vezes a média do intervalo RR.

$$Dur\ (T) = \sum_{i=1}^{n2} Dur(ai) \qquad (13)$$

As fórmulas das condições são :

0,5 Amp*r* (T) < Amp*l* (T) < 1,5 Amp*r*(T) e
0,1RR < Dur (T) < 0,4 RR e
Durt(T) < 0,5 Dur(T) e o ponto de início da onda T < 0,2RR.

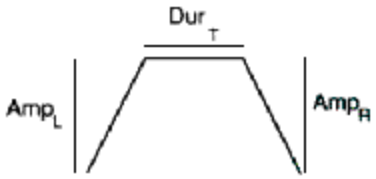


Figura 5.3.2 – Os componentes estruturais de uma onda T: Apml e Ampr são as amplitudes dos lados esquerdo e direito e Durt é a duração do segmento plano do topo

As constantes das fórmulas foram experimentalmente estimadas após testes e possuem forte influência sobre a efetivação do reconhecimento de ondas T.

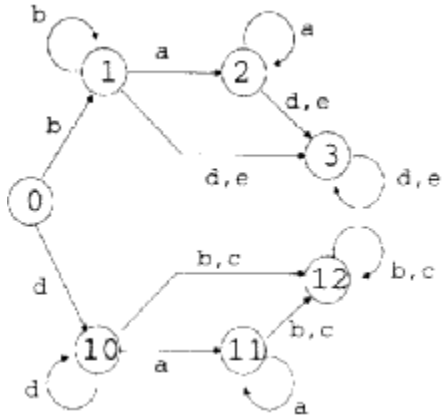


Figura 5.3.3 – Autômato TWAVE que faz a busca por ondas T

O autômato para reconhecimento de ondas P o autômato PWAVE possui a mesma estrutura sintática que o TWAVE , com exceção das condições que são diferentes. No caso de ondas P foi permitida uma maior variação entre as amplitudes dos lados esquerdo e direito da onda. Também, a duração da onda P é uma fração menor da média do intervalo RR do que a duração da onda T. Outro ponto interessante é que uma onda P deve ser reconhecida se ela for a última onda P antes do próximo complexo QRS. Assim, o parsing de uma onda P é feito se não forem reconhecidas ondas P antes do próximo complexo QRS.

Adicionalmente a onda P de topo unitário foi adicionado um autômato DOUBLEPWAVE ao conjunto de autômatos. Sua tarefa é detectar estruturas de ondas P com dois topos que podem ocorrer dentro de um sinal de ECG por causa da diferença do tempo de despolarização do átrio direito e esquerdo.

O autômato QRS tenta detectar um complexo QRS. Sua estrutura sintática é dada pela seguinte expressão regular : (*e*+ *c*+)+ *e** | (*c*+ *e*+)+ *c** e detecta picos íngremes consecutivos. Se for encontrado um candidato sintaticamente correto *c*n1 *e*m1 ... *c*nk *e*mk para um complexo QRS, a duração e amplitude da forma de onda podem ser calculados pelas equações 14 e 15 respectivamente :

$$\text{Dur (QRS)} = \sum_{i=1}^{n1} \text{Dur}(ci) + \sum_{i=1}^{m1} \text{Dur}(ei) + \ldots + \sum_{i=1}^{nk} \text{Dur}(ci) + \sum_{i=1}^{m k} \text{Dur}(ei) \qquad (14)$$

$$\text{Amp(QRS)} = \text{Max}\left\{ \sum_{i=1}^{n1} |\text{Amp}(ci)| \; ; \sum_{i=1}^{m1} |\text{Amp}(ei)| \; ;...; \sum_{i=1}^{nk} |\text{Amp}(ci)| \; ; \sum_{i=1}^{mk} |\text{Amp}(ei)| \right\} \quad (15)$$

O parsing de um complexo QRS é feito se a altura do complexo QRS, Amp (QRS) for maior que seis vezes a amplitude média das primitivas no sinal. Também é necessário que as diferenças de amplitudes entre o ponto de início e o ponto final do QRS possam não ser maior que a amplitude média das primitivas. Isto é uma condição muito importante para rejeitar falsos sinais identificados como QRS que são causados por ruídos.

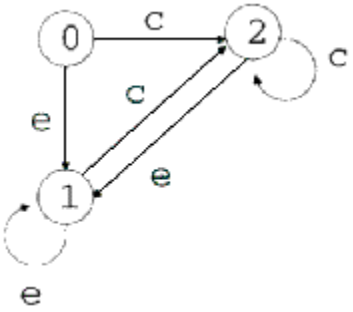


Figura 5.3.4 – Autômato QRS que faz a busca por complexos QRS

Nos casos acima foi feita uma estrutura sintática regular que por si só é suficientemente poderosa para definir precisamente os subpadrões desejados. Precisa-se também calcular características numéricas diferentes dos subpadrões detectados e examinar se eles satisfazem as condições predeterminadas. Só então se pode supor ter encontrado um subpadrão verdadeiro. Foi concluído que a duração e a amplitude de um subpadrão são apropriados para esta finalidade. Foram cuidadosamente evitadas circunstâncias onde devessem ser usados valores absolutos para pontos iniciais e limites. No processamento de sinais digitais biomédicos não se pode fornecer de antemão nenhum limite absoluto porque a escala de medidas varia muito. As considerações são relativas a outras decisões obtidas de sinais de ECG. Primeiro deve ser encontrado um complexo QRS para calcular a média do intervalo RR, que é usado na detecção das ondas P e T. Para a detecção do QRS deve se primeiro calcular a estimativa corrente da amplitude média e duração de uma primitiva no sinal.
Assim não foi usado nenhum parâmetro de ponto inicial dependente de escala no sistema. Porem, este tipo de adaptação de reconhecimento toma um tempo de inicialização que desperdiça geralmente dois ou três ciclos cardíacos do inicio do sinal, que é aceitável em gravações longas de sinais de ECG.

6. IMPLEMENTAÇÃO E TESTES

Afim de prover um dos objetivos específicos deste trabalho, o reconhecimento dos subpadrões de um sinal de ECG, foi desenvolvido um sistema de reconhecimento sintático baseado no sistema proposto por Antti Koski que foi descrito no capítulo 5. Fizemos uma pequena implementação a caráter de testes em linguagem ***Object Pascal*** para avaliar as funcionalidades do sistema. Agregamos a compatibilidade com ECGs DICOM 3.0 utilizando rotinas do software de domínio público EzDicom. Numa segunda fase, o sistema de autômatos será agregado ao sistema **Cyclops Personal**, como um browser para visualização de imagens Dicom nas modalidades ECG e HD, além da funcionalidade extra de classificação dos componentes da forma de onda e seus valores de duração e amplitude. Com a agregação ao cyclops Personal toda a parte de suporte para abertura de arquivos Dicom nestas modalidades será implementada reutilizando-se as rotinas já existentes, o que aumentará a eficiência do sistema principalmente nos procedimentos de abertura e exportação de arquivos Dicom. Numa fase posterior pretende-se também implementar o sistema de autômatos em ***Small Talk*** e adaptá-lo à aplicação **Cyclops Dicom Waveform**, estendendo as funcionalidades deste aplicativo.

6.1 Problemas encontrados durante a implementação

A implementação do sistema de autômatos apresentou alguns problemas na definição das regras de avaliação dos parâmetros para os níveis de aceitação dos subautômatos. O conteúdo contido nas referencias e descrito no capítulo 5 descreve apenas as regras de aceitação dos atributos para o reconhecimento de ondas T. As regras para reconhecimento de complexos QRS apresentam informações insuficientes, e os demais componentes (intervalos e ondas P) não tiveram suas regras de aceitação descritas. Além disso, é apresentado um sistema com dez autômatos, dos quais foram citados apenas oito. Sendo assim, assumiu-se neste trabalho a utilização de pelo menos sete autômatos para desempenhar a análise do sinal e foram formuladas algumas regras de aceitação arbitrárias, seguindo observações de sinais coletados.

6.2 Implementação

O sistema foi dividido em três partes: módulo GUI, módulo de classificação e sistema de autômatos.

6.2.1 O módulo GUI

O objetivo deste módulo é prover uma interface de usuário para a manipulação de imagens Dicom na modalidade ECG como parte do sistema Cyclops Personal desenvolvido pelo Cyclops Project, visto que tal sistema ainda não possuía suporte para a visualização de arquivos na modalidade ECG/Hemodinâmica. O módulo GUI contém as rotinas para impressão e ajuste do sinal conforme suas várias formas de visualização. As rotinas para a leitura de

imagens Dicom na modalidade ECG foram implementadas aproveitando o suporte fornecido pelo sistema Cyclops Personal estendendo algumas funções já implementadas. O módulo GUI também fornece suporte para a visualização dos demais parâmetros do arquivo contidos nos campos *patient*, *study*, *series information* e *equipment* que armazenam informações sobre as características do exame como nome do paciente, data da amostra etc, e as características do sinal coletado, como a taxa de amostragem, número de canais, fabricante do equipamento, operador do eletrocardiógrafo etc. Os valores dos campos devem ser repassados pela rotina de abertura do arquivo como um objeto contendo os campos e uma lista contendo os canais do ECG afim de inicializar campos do módulo GUI e permitir a manipulação e a classificação do sinal.

6.2.2 Módulo de classificação

O módulo de classificação serve como uma interface entre o módulo GUI e o sistema de autômatos. Este módulo provê rotinas para a carga das listas de cada canal e as funções de classificação do sinal. Este módulo foi provido pela classe TClassificador que contém todas as rotinas necessárias para a geração dos labels e o autômato ECG para a realização do parsing das primitivas geradas.

Após a abertura de um ECG, o módulo GUI repassa um conjunto de listas contendo os valores de cada canal para o módulo de classificação.

Para a classificação do sinal é escolhida (pelo usuário) uma das doze listas para ser utilizada no processo de analise.

O sinal é classificado em três passos:

1 Amostragem utilizando a lista escolhida - Esta função é provida pela função *colete_amostras* na classe Tclassificador. A amostragem é feita sobre a lista escolhida saltos de 4 posições e são gerados os labels em função do valor da inclinação do sinal. Os valores coletados são armazenados em uma lista de primitivas, contendo os valores de amplitude *Amp_i*, duração, *Dur_i*, label, *label_i* e o valor nulo em *type_wave_i* em cada posição.

2 Classificação dos dados amostrados - Esta função é provida pelo procedimento *parseAVF* que chama o procedimento *parser* da classe TAutomatoECG e faz com que a lista de primitivas gerada no passo 1 seja classificada pelo processo de análise iniciado pelo ECG.

3 Generalização dos valores - Esta função é provida pelo procedimento *distribua_padroes.* Nesta rotina, todas as doze listas DRG de cada canal atualizam os valores de type_wave_i conforme os valores da lista de primitivas classificada.

Observações

- Quanto a amostragem - verificou-se que o melhor valor para amostragem do sinal seria um valor a cada quatro posições. Como as amostras utilizadas nos testes foram coletadas a uma taxa de amostragem de 240 Hz, o sinal passa a ser amostrado em 60 Hz. O sistema proposto em [2] utilizou um aparelho com taxa de amostragem de 400 Hz com valores de saltos a cada 15 posições.

- Geração dos labels - Observou-se que a geração dos labels seguindo a taxa de amostragem com saltos de quatro posições gerava vários erros e adotou-se um valor que desse uma menor margem para a geração de labels 'a' (segmento plano) diminuindo o intervalo de [-20 .. 20] para [-10..10]:

```
// Alteração no intervalo do label 'a' - > [-10 .. 10]
fat := 35;
ang := arcTan2((y2 - y1),((x2 - x1)*fat));
ang := RadToDeg(ang);
if ((ang <= 90) and (ang > 75))then car := 'c'
else
 if ((ang <= 75) and (ang > 10))then car := 'b' // > 20
 else
  if ((ang <= 10) and (ang >-10))then car := 'a'// <= 20
  else
   if((ang <=-10) and (ang >-75))then car := 'd'
   else
    if((ang <= -75) and (ang >-90)) then car := 'e';
 result := car;
```

Além disso, para calcular o valor da inclinação da reta, os valores da diferença de tempo dx precisaram ser normalizados por um fator cujo melhor valor ficou entre 35.

6.2.3 Implementação do sistema de autômatos finitos

Para a classificação do sinal foram criados seis autômatos específicos para o reconhecimento de cada subpadrão. O sétimo autômato, o autômato ECG é mais uma espécie de analisador léxico. A primeira função do autômato ECG é coletar o tempo de um intervalo RR para iniciar o processo de parsing, tempo este que servirá de parâmetro para a avaliação da aceitação dos subpadrões na chamada dos subautômatos. A outra função é controlar o processo de análise ativando os autômatos em função do label presente na entrada e do seu estado atual.

Quanto a implementação dos autômatos, seguiu-se o paradigma proposto na cadeira de linguagens formais: cada autômato, com exceção do autômato ECG foi implementado na forma genérica contendo uma tabela de transição específica. A única modificação nesta implementação foi feita no algoritmo de avaliação das sentenças, onde a regra de reconhecimento de um prefixo é feita em função do estado atual e da satisfação das regras de aceitação (que avaliam os atributos anexados aos tokens), como pode ser visto no seguinte trecho do código:

```
while continuar do
begin
  est := transicao(est,car);
  if  (avaliaParametros(Amp_l,Amp_r, Dur, Dur_T, _RR))
  or  (  VEF[est] = 1 )                                  then

  { se a avaliação dos parametros for aceita ou
    se for encontrado o estado final o automato termina a busca}

    continuar := False
  else
    ... {atualiza valores de amp / tempo}

end; // while
Amp := (Amp_l + Amp_r)/2;
if est <> 8 then
begin
  result := True;
end else
  result := False;
```

Como o algoritmo de busca pode parar sem alcançar um estado final, o autômato adquire o poder de reconhecer um prefixo da linguagem, sem ter de analisar toda a sentença. Isto quer dizer que, para reconhecer uma determinada onda, o autômato construído para tal continua trabalhando enquanto onda ainda não apresentar seus valores de duração, amplitude e posicionamento no ciclo cardíaco dentro das condições impostas nas regras de avaliação dos atributos.

Uma outra modificação deste sistema em relação ao sistema proposto em [2] foi feito quanto a definição das máquinas de estado, pois observou-se que se fosse seguido os diagramas propostos e descritos no capitulo 5, alguns prefixos

indesejáveis poderiam ser reconhecidos. Sendo assim, foram anexados estados de erro em todos os autômatos para corrigir eventuais erros de parsing e estados finais para aumentar a eficiência da busca.

6.2.3.1 Especificação das regras de aceitação dos autômatos

O autômato T segue exatamente as regras de aceitação propostas em [2] e descritas no capitulo 5:

0,5 Amp*r* (T) < Amp*l* (T) < 1,5 Amp*r*(T) e
0,1RR < Dur (T) < 0,4 RR e
Durt(T) < 0,5 Dur(T) e o ponto de início da onda T < 0,2RR.

A estrutura sintática é definida pela máquina de estados do autômato T. Esta apresenta algumas modificações em relação ao modelo original descrito no capitulo 5, pois houve a necessidade de incluir estados finais afim de auxiliar o processo de parsing conforme a figura 6.2.3.1.1:

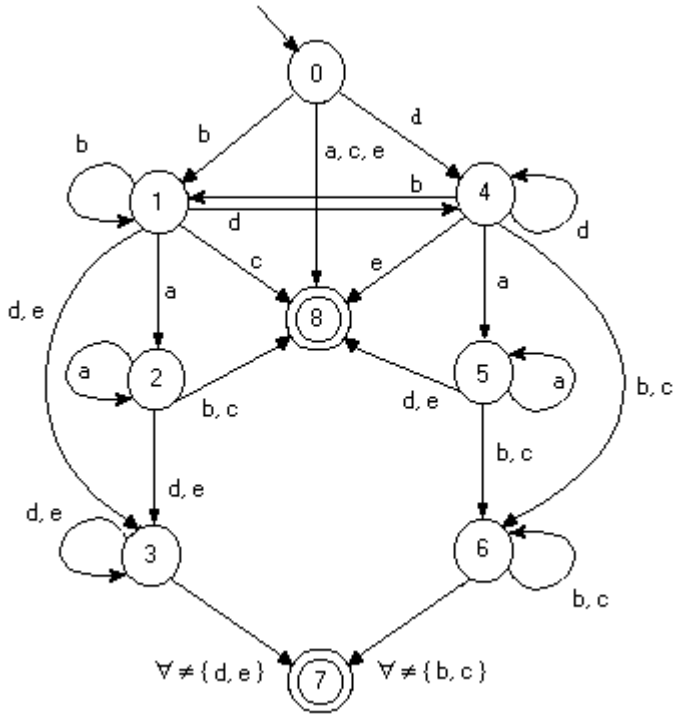


Figura 6.2.3.1.1 - Máquina de estados do autômato T, com a inclusão de estados finais.

Os autômatos ST, TP e PQ não possuem regras de aceitação da avaliação dos atributos, apenas reconhecem um segmento plano no sinal. Porém, regras de aceitação levando em conta a amplitude média do segmento plano podem ser agregadas ao sistema afim de aumentar o grau de liberdade na classificação dos segmentos. Com isto poderia ser utilizado um autômato que permitisse outros labels diferentes de *'a'* , o que corrigiria alguns erros de classificação encontrados nos testes.

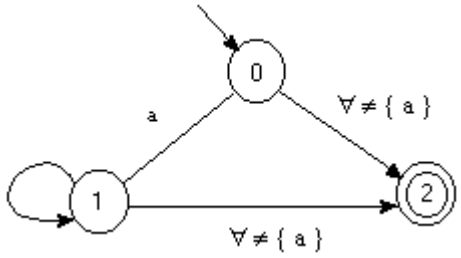


Figura 6.2.3.1.2 – Máquina de estados dos autômatos ST, TP e PQ.

Quanto ao autômato QRS foram descritas regras de avaliação dos parâmetros de amplitude mas os testes mostraram que a estrutura sintática descrita nos valores dos labels é suficiente para reconhecer a linguagem definida por um complexo QRS. Assim, também não foram incluídas regras de avaliação dos atributos nos complexos QRS. A única modificação feita em relação ao modelo original foi a adição de um estado de erro para filtrar labels que não fazem parte da linguagem conforme a figura 6.2.3.1.3:

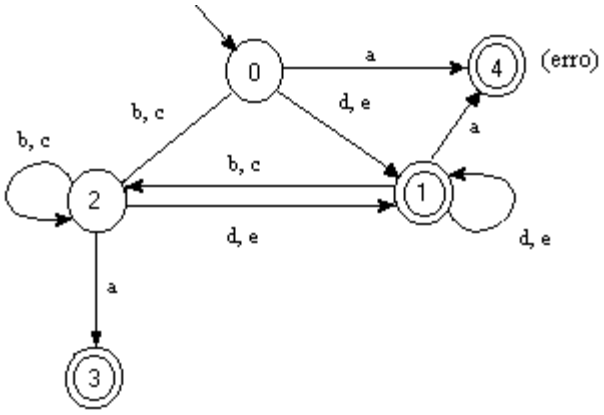


Figura 6.2.3.1.3 – Máquina de estados do autômato QRS, com a inclusão de um estado de erro.

Para o autômato P foram definidas regras de avaliação seguindo a observação da forma e duração das ondas P. A estrutura sintática definida pela máquina de estados do autômato P é suficiente para reconhecer ondas P de um único pico. No sistema original havia um autômato DoublePwave para desempenhar a função de reconhecimento de ondas P com dois picos, mas o falta de referencias sobre os parâmetros deste novo componente impediu a implementação de tal autômato, além da falta de amostras de sinais contendo ondas P com dois picos.
As regras de avaliação dos atributos foram determinadas tomando como referencia a duração do intervalo RR, quanto ao ponto de início e o ponto de finalização da onda P dentro deste intervalo:

$$0.1\ RR < \text{Duração da onda P} < 0.11\ RR$$

A estrutura sintática do autômato P segue a mesma estrutura do autômato T, assim a máquina de estados do autômato P é exatamente a mesma do autômato T.

6.3 Testes

Para testar a funcionalidade do sistema de autômatos, foi utilizado um eletrocardiograma codificado em Dicom na modalidade *12Lead ECG*. Este arquivo fornece amostras coletadas simultâneamente de todos os doze canais do exame.

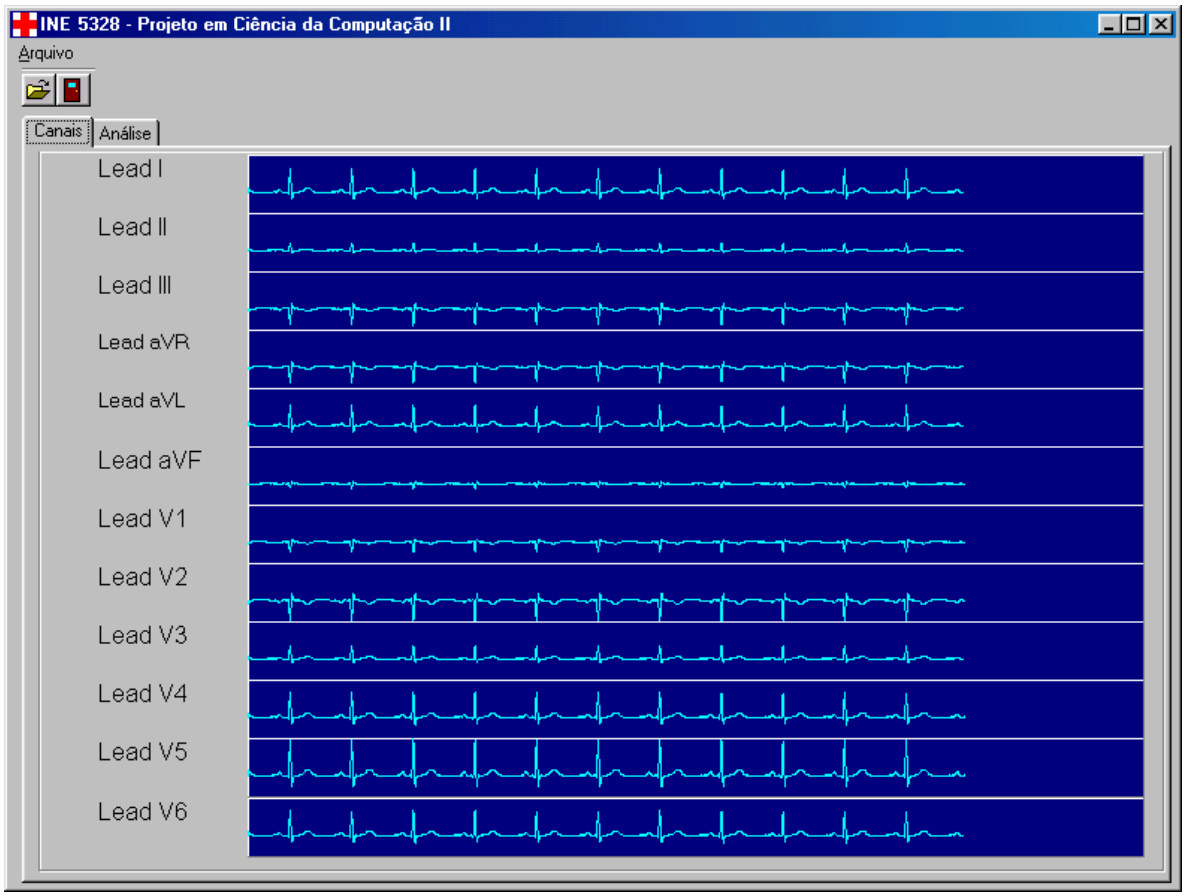


Figura 6.3.1 – Visualização de um arquivo DICOM com doze canais.

Apesar de o sistema de autômatos suportar a análise de ondas invertidas, apenas os canais ***Lead I***, ***Lead AVL***, ***V4***, ***V 5*** e ***V6*** apresentaram um bom resultado na classificação. Dentre estes, o canal analisado com melhores resultados foi o ***Lead AVL*** , por causa da deficiência do cálculo do intervalo RR e por causa da potência do sinal nas outras derivações, o que alterou o codificação das sentenças geradas, resultando em sentenças que não fazem parte da linguagem definida para os subpadrões.

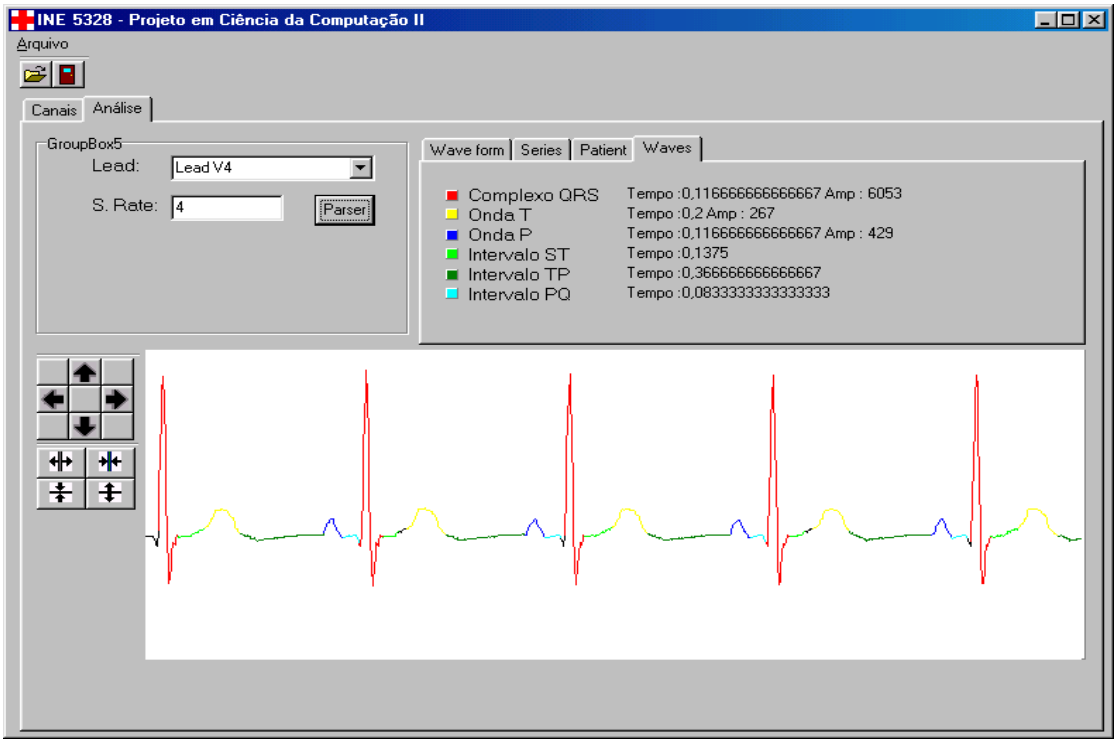


Figura 6.3.2 – Lead V4 classificado.

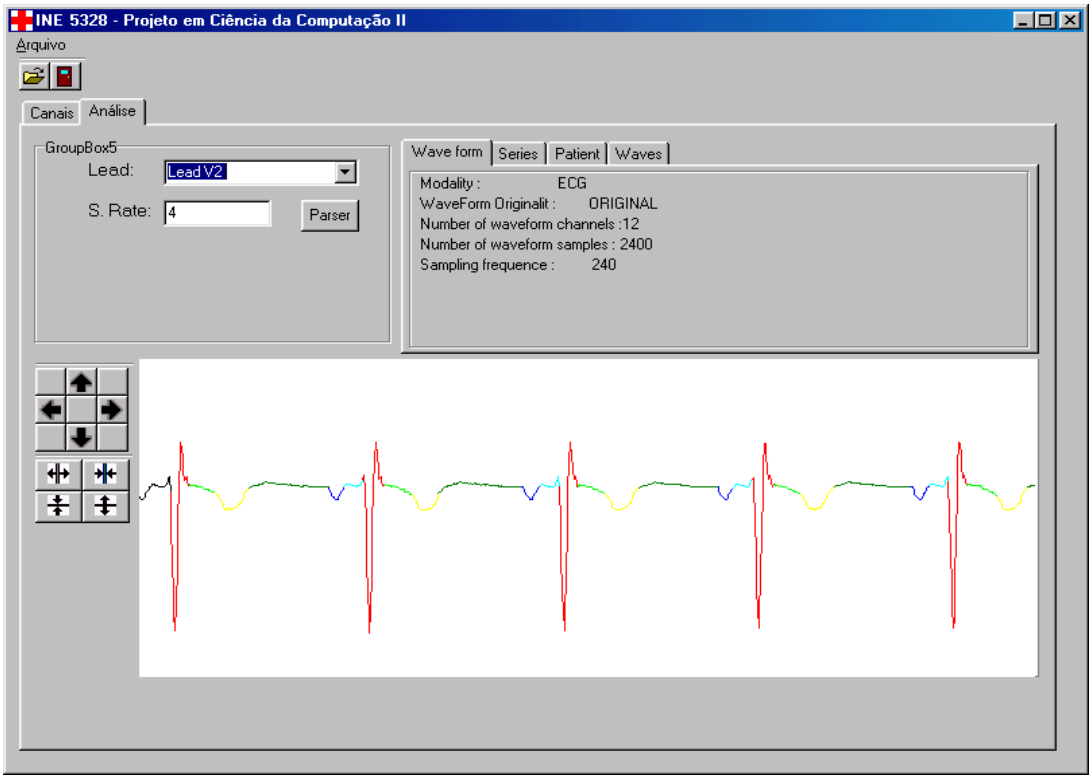


Figura 6.3.3 – Lead V2 classificado.

Um fator importante é o posicionamento do ponto de início dos componentes do sinal, que varia em função da taxa de amostragem escolhida para a geração dos labels, assim quanto maior a taxa de amostragem maior será o atraso no ponto de início e de finalização das ondas/ segmentos.

6.4 Considerações finais

Devido ao pequeno número de amostras disponíveis para se testar as funcionalidades do sistema, esta implementação se mostrou eficiente apenas para a classificação de sinais bem comportados.

7. CONCLUSÕES E TRABALHOS FUTUROS

Através deste trabalho pode-se fazer uma avaliação a nível de protótipo do sistema de autômatos provando ser uma boa abordagem como técnica de reconhecimento de eletrocardiogramas. Apesar da falta de amostras disponíveis para uma fase de testes consistente, o sistema implementado mostrou-se eficiente na tarefa de análise e classificação de sinais bem comportados. Além disso pode ter suas funcionalidades estendidas de maneira fácil. Para sinais contendo algum tipo de alteração que possa ser padronizada, pode ser implementado um novo conjunto de autômatos que apresente maiores graus de liberdade na avaliação dos parâmetros e que seja ápto para reconhecer linguagens que representem subpadrões modificados. Neste caso seria implementado um conjunto de autômatos específico para cada caso de anormalia. Uma aplicação difícil seria a classificação de amostras de pacientes acometidos por anormalias como fibrilação atrial conforme a figura 7.1, pois o comprimento do intervalo RR é decrescente com o tempo. Como o valor do intervalo RR é essencial para a avaliação dos parâmetros de duração do sinal nesta implementação, isto seria praticamente impossível se for usada a estrutura atual do sistema.

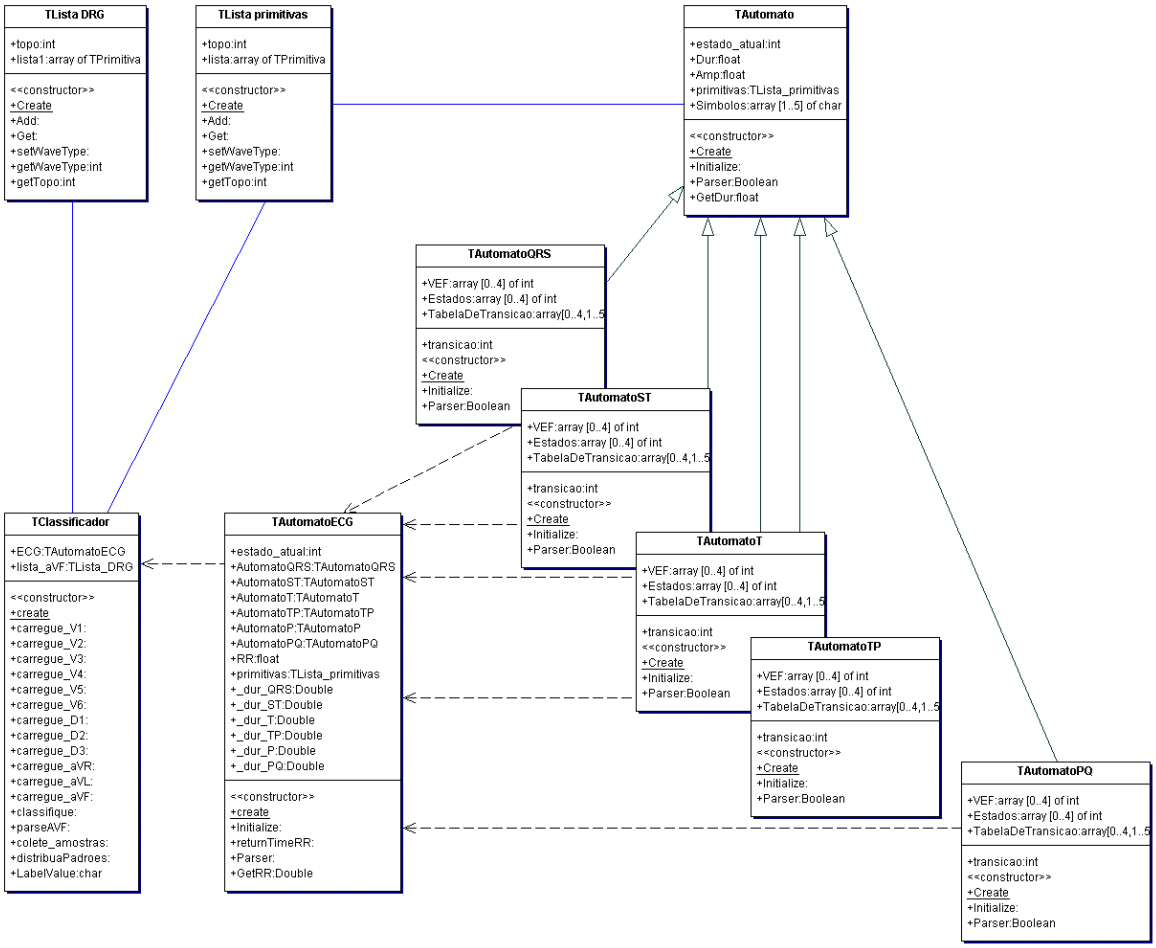
Figura 7.1 - Amostra de ECG apresentando fibrilação atrial.

Outro ponto importante abordado neste trabalho foi a implementação de uma estrutura para visualização de ECGs digitalizados em formato Dicom 3.0 o que é de grande utilidade, principalmente quanto ao auxílio à visualização do sinal e a facilidade de intercâmbio de informações entre profissionais.

Em trabalhos futuros a estrutura do sistema para reconhecimento dos componentes do eletrocardiograma pode ser aproveitada como um pré-processamento para a análise do sinal. Opós conseguir colher informações sobre as durações e amplitudes de cada componente e em cada canal, estas poderiam ser utilizadas como parâmetros de um sistema capaz de detectar anomalias previamente classificadas. Assim, poderia ser feito um estudo sobre as características do sinal utilizando outras técnicas de reconhecimento de padrões mais indicadas para este fim como MEDIDAS DE DISTÂNCIA que eficientemente serviriam para a classificação de anormalidades aumentando ainda mais as funcionalidades do sistema.

8. ANEXOS

8.1 Anexo 1 – Diagrama de classes



8.2 Anexo 2 – Sistema de autômatos

8.2.1 – Maquinas de estado

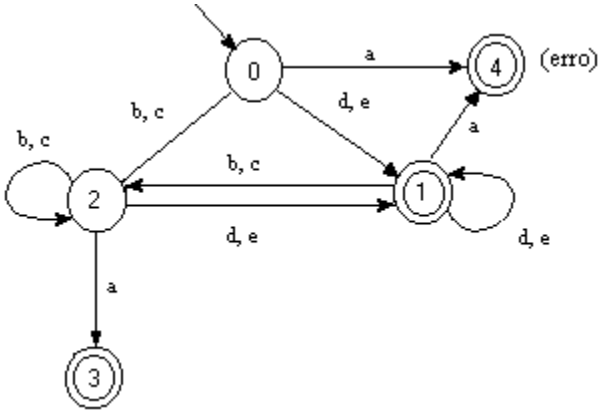


Figura 8.2.1.1 – Máquina de estados para o autômato QRS.

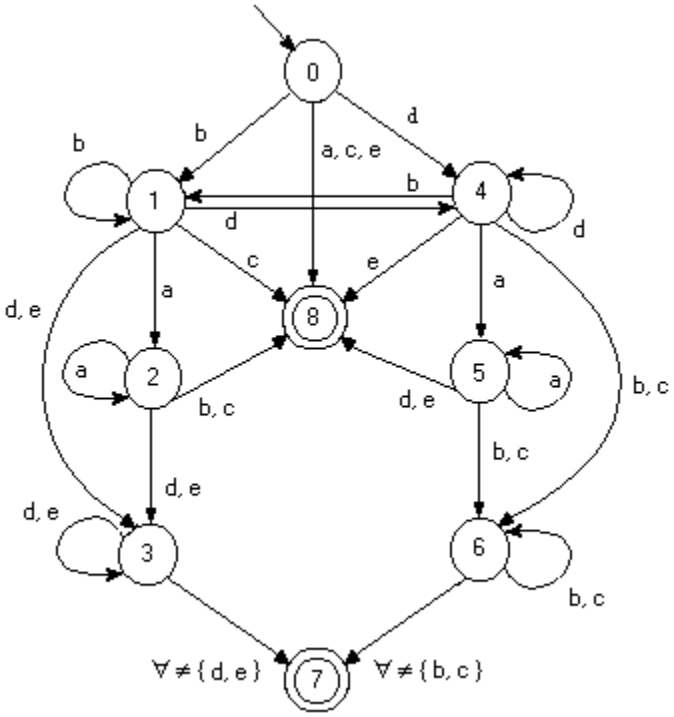


Figura 8.2.1.2 – Máquina de estados para os autômatos P eT.

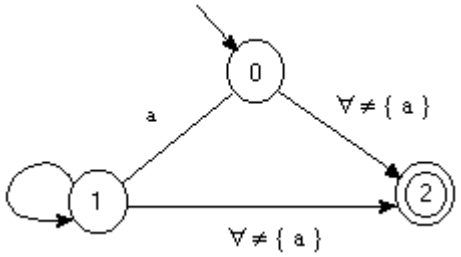


Figura 8.2.1.3 – Máquina de estados para autômatos ST, TP e PQ.

8.2.2 – Tabelas de transição

8.2.2.1 Tabela de Transição Autômato QRS

Estado/ Label	a	b	c	d	e
0	4	2	2	1	1
1	4	2	2	1	1
2	3	2	2	1	1
*3	-	-	-	-	-
*4	-	-	-	-	-

8.2.2.2 Tabela de Transição Autômatos P e T

Estado/ Label	a	b	c	d	e
0	8	1	8	4	8
1	2	1	8	3	3
2	2	8	8	3	3
3	7	7	7	3	3
4	5	6	6	4	8
5	5	6	6	8	8
6	7	6	6	7	7
*7	-	-	-	-	-
*8	-	-	-	-	-

8.2.2.3 Tabela de Transição para Autômatos ST, TP e PQ

Estado/ Label	a	b	c	d	e
0	1	4	4	4	4
1	2	4	4	4	4
2	2	3	3	3	3
*3	-	-	-	-	-
*4	-	-	-	-	-

8.2.2.4 Tabela de Transição Autômato ECG

Estado/ Label	a	b	C	d	e
0	ST(1)	T(2)	Q(6)	T(2)	Q(6)
1	-	T(2)	Q(6)	T(2)	Q(6)
2	TP(3)	P(4)	Q(6)	P(4)	Q(6)
3	-	P(4)	Q(6)	P(4)	Q(6)
4	PQ(5)	-	Q(6)	-	Q(6)
5	-	-	Q(6)	-	Q(6)
6	-	-	-	-	-

8.3 Anexo 3 - Fontes

8.3.1 – Classe TAutomato

```
unit uAutomato;

interface
uses uLista_primitivas;
type TAutomato = class
protected
     estado_atual : integer;
     Dur, Amp     : Double;
public
     primitivas : TLista_primitivas;
     Simbolos: array [1..5] of char;
     constructor Create(_primitivas : TLista_primitivas);   virtual;
     procedure Initialize; virtual;
     function Parser( var Pos : integer) : Boolean;         virtual;
     function GetDur : double;
     function GetAmp : double;
end;

implementation

constructor TAutomato.Create(_primitivas : TLista_primitivas);
begin
  (*inicialização do vetor de simbolos*)
  Simbolos[1]:='a';
  Simbolos[2]:='b';
  Simbolos[3]:='c';
  Simbolos[4]:='d';
  Simbolos[5]:='e';
  Dur        := 0;
  Amp        := 0;
  primitivas := _primitivas;
end;

function TAutomato.GetAmp: double;
begin
  Result := Self.Amp;
end;
```

```
function TAutomato.GetDur: double;
begin
  Result := Self.Dur;
end;

procedure TAutomato.Initialize;
begin
  Dur        := 0;
  Amp        := 0;
end;

function TAutomato.Parser(var Pos: integer): Boolean;
begin
end;
end.
```

8.3.2 – Classe Tlista_Primitivas

```
unit uLista_primitivas;

interface
 type TPrimitiva = record
   name : char;
   Dur, Amp,x,y : Double;
   waveType : integer;
 end;
 type TDRG = record
   Amp : Double;
   waveType : integer;
 end;

 type TLista_primitivas = class
   protected
     lista : array of TPrimitiva;
     topo : integer;
   public
     constructor Create;
     procedure initialize();
     procedure Add(_name : char; _Dur, _Amp,_x,_y : Double);
     procedure Get(index : integer; var vname : char; var vDur, vAmp :
Double);
     procedure setWaveType(_Pos, _type : integer);
     function getWaveType(_Pos : integer; var _x,_y : Double) : integer;
     function getTopo : integer;
 end;

 type TLista_DRG = class
   protected
     lista : array of TDRG;
     topo : integer;
   public
     constructor Create;
     procedure initialize();
     procedure Add(_Amp : Double);
     procedure Get(index : integer; var vAmp : Double);
     procedure setWaveType(_Pos, _type : integer);
```

```
     function getWaveType(_Pos : integer) : integer;
     function getTopo : integer;
 end;

implementation

{ Tlista_primitivas }
```

```
procedure Tlista_primitivas.Add(_name: char; _Dur, _Amp, _x, _y :
Double);
begin
  SetLength(lista,Length(lista)+1);
  lista[Length(lista)-1].name := _name;
  lista[Length(lista)-1].Dur  := _Dur;
  lista[Length(lista)-1].Amp  := _Amp;
  lista[Length(lista)-1].x    := _x;
  lista[Length(lista)-1].y    := _y;

  Inc(topo);
end;

constructor Tlista_primitivas.Create;
begin
  initialize;
end;

procedure Tlista_primitivas.Get(index: integer; var vname: char; var
vDur,
  vAmp: Double);
begin
  vname := lista[index].name;
  vDur  := lista[index].Dur;
  vAmp  := lista[index].Amp;
end;

function Tlista_primitivas.getTopo: integer;
begin
  Result := Length(lista)-1;
end;

function TLista_primitivas.getWaveType(_Pos: integer;  var _x,_y :
Double): integer;
begin
  _x     := lista[_Pos].x;
  _y     := lista[_Pos].y;
  result := lista[_Pos].waveType;
end;

procedure Tlista_primitivas.initialize;
```

```
begin
  topo := -1;
  SetLength(lista,0);
end;

procedure TLista_primitivas.setWaveType(_Pos, _type: integer);
begin
  lista[_Pos].waveType := _type;
end;


{ TLista_DRG }

procedure TLista_DRG.Add(_Amp: Double);
begin
  SetLength(lista,Length(lista)+1);
  lista[Length(lista)-1].Amp  := _Amp;
  Inc(topo);
end;

constructor TLista_DRG.Create;
begin
  initialize;
end;

procedure TLista_DRG.Get(index: integer; var vAmp: Double);
begin
  vAmp  := lista[index].Amp;
end;

function TLista_DRG.getTopo: integer;
begin
  Result := Length(lista)-1;
end;

function TLista_DRG.getWaveType(_Pos: integer): integer;
begin
  result := lista[_Pos].waveType;
end;

procedure TLista_DRG.initialize;
begin
  topo := -1;
  SetLength(lista,0);
end;

procedure TLista_DRG.setWaveType(_Pos, _type: integer);
begin
  lista[_Pos].waveType := _type;
end;

end.
```

8.3.3 – Classe TAutomatoECG

```
unit uAutomatoECG;

interface
 uses uLista_primitivas, uAutomatoQRS, uAutomatoST, uAutomatoT,
uAutomatoTP,
 uAutomatoP, uAutomatoPQ;

 type TAutomatoECG = class
   protected
     estado_atual : integer;
     AutomatoQRS  : TAutomatoQRS;
     AutomatoST   : TAutomatoST;
     AutomatoT    : TAutomatoT;
     AutomatoTP   : TAutomatoTP;
     AutomatoP    : TAutomatoP;
     AutomatoPQ   : TAutomatoPQ;
     RR           : Double;

   public
     primitivas : TLista_primitivas;
     _dur_QRS : Double;
     _dur_ST  : Double;
     _dur_T   : Double;
     _dur_TP  : Double;
     _dur_P   : Double;
     _dur_PQ  : Double;
     _Amp_QRS : Double;
     _Amp_T   : Double;
     _Amp_P   : Double;
     constructor Create;
     destructor Destroy;
     procedure Initialize;
     procedure returnTimeRR(var Pos : Integer ;_rate : Integer);
     procedure Parser(var Pos : integer; _RR : Double; _rate : Integer);
     function GetRR : Double;
 end;

implementation

{ TAutomatoECG }

constructor TAutomatoECG.Create;
begin
  estado_atual := 0;
  primitivas  := TLista_primitivas.Create;
  AutomatoQRS := TAutomatoQRS.Create(primitivas);
  AutomatoST  := TAutomatoST.Create(primitivas);
  AutomatoT   := TAutomatoT.Create(primitivas);
  AutomatoTP  := TAutomatoTP.Create(primitivas);
  AutomatoP   := TAutomatoP.Create(primitivas);
  AutomatoPQ  := TAutomatoPQ.Create(primitivas);
end;

destructor TAutomatoECG.Destroy;
begin
  AutomatoQRS.Free;
  AutomatoST.Free;
```

```
  AutomatoT.Free;
  AutomatoTP.Free;
  AutomatoP.Free;
  AutomatoPQ.Free;
end;

function TAutomatoECG.GetRR: Double;
begin
  Result := Self.RR;
end;

procedure TAutomatoECG.Initialize;
begin
  estado_atual := 0;
end;

procedure TAutomatoECG.returnTimeRR(var Pos : Integer ;_rate : Integer);
var
  i, pos_ant : integer;
  dur_RR,Amp_QRS, Dur_i, Amp_i : Double;
  car : Char;
  atualize : Boolean;
begin
  primitivas.Get(Pos,car,Dur_i,Amp_i);
  dur_RR := 0;
  while (Pos < primitivas.getTopo - 1)and(estado_atual <> 2) do
  begin
    pos_ant := pos + 1;
    atualize := False;
    case car of
    'c' : begin
            case estado_atual of
            0  :  if AutomatoQRS.Parser(Pos) then
                    estado_atual := 1
                  else
                    begin
                       for i := pos_ant - 1 to Pos do
                         primitivas.setWaveType(i,30);
                       Pos := pos_ant;
                    end;

            1  :  if AutomatoQRS.Parser(Pos) then
                  begin
                    estado_atual := 2;
                    dur_RR := dur_RR + AutomatoQRS.GetDur;
                  end else
                    begin
                       for i := pos_ant - 1 to Pos do
                         primitivas.setWaveType(i,30);
                       Pos := pos_ant;
                    end;

            else
                    atualize := True;

            end; // case estado...
          end; // c: ...
```

```
    'e' : begin
            case estado_atual of
            0  :  if AutomatoQRS.Parser(Pos) then
                    estado_atual := 1
                  else
                    begin
                       for i := pos_ant - 1 to Pos do
                         primitivas.setWaveType(i,30);
                       Pos := pos_ant;
                    end;

            1  :  if AutomatoQRS.Parser(Pos) then
                  begin
                    estado_atual := 2;
                    dur_RR := dur_RR + AutomatoQRS.GetDur;
                  end else
                    begin
                       for i := pos_ant - 1 to Pos do
                         primitivas.setWaveType(i,30);
                       Pos := pos_ant;
                    end;

            else
                    atualize := True;

            end; // case estado...
          end; // e: ...
    else
          atualize := True;

    end; // case

    if (((Pos  + 1) <= primitivas.getTopo - 1)and atualize) then
      Inc(Pos);
    if estado_atual = 1 then
       dur_RR := dur_RR + (1/240);

    primitivas.Get(Pos, car, Dur_i, Amp_i);
  end; //while
  RR := dur_RR * _rate;
end;

procedure TAutomatoECG.Parser(var Pos : integer; _RR : Double; _rate :
Integer);
var
  i, pos_ant, pos_def : integer;
  dur_QRS, dur_ST, dur_T, dur_TP, dur_P, dur_PQ,
  Amp_QRS, Amp_T, Amp_P, Dur_i, Amp_i : Double;
  car : Char;
  atualize : Boolean;
  tempo_atual : Double;
begin
  dur_QRS := 0;
  dur_ST  := 0;
  dur_T   := 0;
  dur_TP  := 0;
```

```
dur_P   := 0;
dur_PQ  := 0;
Amp_QRS := 0;
Amp_T   := 0;
Amp_P   := 0;
tempo_atual := 0;
pos_def := Pos;

primitivas.Get(Pos,car,Dur_i,Amp_i);
tempo_atual := ((1/240*4)* (Pos-pos_def + 1));
while (Pos < primitivas.getTopo - 1)and(estado_atual <> 6) do
begin
  pos_ant := pos + 1;
  atualize := False;
  case car of
  'a' : begin
          case estado_atual of
          0  :  if AutomatoST.Parser(Pos) then
                begin
                  estado_atual := 1;
                  dur_ST       := AutomatoST.GetDur;
                end else
                begin
                  for i := pos_ant - 1 to Pos do
                    primitivas.setWaveType(i,30);
                  Pos := pos_ant;
                end;

          2  :  if AutomatoTP.Parser(Pos) then
                begin
                  estado_atual := 3;
                  dur_TP       := AutomatoTP.GetDur;
                end else
                begin
                  for i := pos_ant - 1 to Pos do
                    primitivas.setWaveType(i,30);
                  Pos := pos_ant;
                end;

          4  :  if AutomatoPQ.Parser(Pos) then
                begin
                  estado_atual := 5;
                  dur_PQ       := AutomatoPQ.GetDur;
                end else
                begin
                  for i := pos_ant - 1 to Pos do
                    primitivas.setWaveType(i,30);
                  Pos := pos_ant;
                end;
          else
                  atualize := True;
          end; // case
        end;
  'b' : begin
          case estado_atual of
       0  : if (tempo_atual < 0.2*_RR) and (tempo_atual > 0.06*RR) then
                begin
```

```
          if AutomatoT.Parser(Pos, _RR) then
          begin
            estado_atual := 2;
            dur_T      := AutomatoT.GetDur;
            Amp_T      := AutomatoT.GetAmp;
          end else
          begin
            for i := pos_ant - 1 to Pos do
              primitivas.setWaveType(i,30);
            Pos := pos_ant;
          end;
        end else Pos := pos_ant;


 1  : if (tempo_atual < 0.2*_RR) and (tempo_atual > 0.06*RR) then
        begin
          if AutomatoT.Parser(Pos, _RR) then
          begin
            estado_atual := 2;
            dur_T      := AutomatoT.GetDur;
            Amp_T      := AutomatoT.GetAmp;
          end else
          begin
            for i := pos_ant - 1 to Pos do
              primitivas.setWaveType(i,30);
            Pos := pos_ant;
          end;
        end else Pos := pos_ant;

2  : if (tempo_atual > 0.65*_RR) and (tempo_atual < 0.8*_RR) then
          if AutomatoP.Parser(Pos, _RR) then
          begin
            estado_atual := 4;
            dur_P      := AutomatoP.GetDur;
            Amp_P      := AutomatoP.GetAmp;
          end else
          begin
            for i := pos_ant - 1 to Pos do
              primitivas.setWaveType(i,30);
            Pos := pos_ant;
          end;

3  : if (tempo_atual > 0.65*_RR) and (tempo_atual < 0.8*_RR) then
        begin
          if AutomatoP.Parser(Pos, _RR) then
          begin
            estado_atual := 4;
            dur_P        := AutomatoP.GetDur;
            Amp_P        := AutomatoP.GetAmp;
          end else
          begin
            for i := pos_ant - 1 to Pos do
              primitivas.setWaveType(i,30);
            Pos := pos_ant;
          end;
        end else Pos := pos_ant;
    else
```

```
                atualize := True;

        end; // case
      end;
'c' : begin
        case estado_atual of
        0  :  if AutomatoQRS.Parser(Pos) then
              begin
                estado_atual := 6;
                dur_QRS      := AutomatoQRS.GetDur;
                Amp_QRS      := AutomatoQRS.GetAmp;
              end else
              begin
                for i := pos_ant - 1 to Pos do
                  primitivas.setWaveType(i,30);
                Pos := pos_ant;
              end;

        1  :  if AutomatoQRS.Parser(Pos) then
              begin
                estado_atual := 6;
                dur_QRS      := AutomatoQRS.GetDur;
                Amp_QRS      := AutomatoQRS.GetAmp;
              end else
              begin
                for i := pos_ant - 1 to Pos do
                  primitivas.setWaveType(i,30);
                Pos := pos_ant;
              end;

        2  :  if AutomatoQRS.Parser(Pos) then
              begin
                estado_atual := 6;
                dur_QRS      := AutomatoQRS.GetDur;
                Amp_QRS      := AutomatoQRS.GetAmp;
              end else
              begin
                for i := pos_ant - 1 to Pos do
                  primitivas.setWaveType(i,30);
                Pos := pos_ant;
              end;

        3  :  if AutomatoQRS.Parser(Pos) then
              begin
                estado_atual := 6;
                dur_QRS      := AutomatoQRS.GetDur;
                Amp_QRS      := AutomatoQRS.GetAmp;
              end else
              begin
                for i := pos_ant - 1 to Pos do
                  primitivas.setWaveType(i,30);
                Pos := pos_ant;
              end;

        4  :  if AutomatoQRS.Parser(Pos) then
              begin
                estado_atual := 6;
```

```
              dur_QRS      := AutomatoQRS.GetDur;
              Amp_QRS      := AutomatoQRS.GetAmp;
            end else
            begin
              for i := pos_ant - 1 to Pos do
                primitivas.setWaveType(i,30);
              Pos := pos_ant;
            end;

       5  :  if AutomatoQRS.Parser(Pos) then
            begin
              estado_atual := 6;
              dur_QRS      := AutomatoQRS.GetDur;
              Amp_QRS      := AutomatoQRS.GetAmp;
            end else
            begin
              for i := pos_ant - 1 to Pos do
                primitivas.setWaveType(i,30);
              Pos := pos_ant;
            end;
       else
              atualize := True;

       end; // case

      end;
'd' : begin
       case estado_atual of
     0  : if (tempo_atual < 0.2*_RR) and (tempo_atual > 0.06*RR) then
            if AutomatoT.Parser(Pos, _RR) then
            begin
              estado_atual := 2;
              dur_T      := AutomatoT.GetDur;
              Amp_T      := AutomatoT.GetAmp;
            end else
            begin
              for i := pos_ant - 1 to Pos do
                primitivas.setWaveType(i,30);
              Pos := pos_ant;
            end;

     1  : if (tempo_atual < 0.2*_RR) and (tempo_atual > 0.06*RR) then
           begin
            if AutomatoT.Parser(Pos, _RR) then
            begin
              estado_atual := 2;
              dur_T      := AutomatoT.GetDur;
              Amp_T      := AutomatoT.GetAmp;
            end else
            begin
              for i := pos_ant - 1 to Pos do
                primitivas.setWaveType(i,30);
              Pos := pos_ant;
            end;
           end else Pos := pos_ant;

    2  : if (tempo_atual > 0.65*_RR) and (tempo_atual < 0.8*_RR) then
```

```
              if AutomatoP.Parser(Pos, _RR) then
              begin
                estado_atual := 4;
                dur_P      := AutomatoP.GetDur;
                Amp_P      := AutomatoP.GetAmp;
              end else
              begin
                for i := pos_ant - 1 to Pos do
                  primitivas.setWaveType(i,30);
                Pos := pos_ant;
              end;

    3  : if (tempo_atual > 0.65*_RR) and (tempo_atual < 0.8*_RR) then
              begin
              if AutomatoP.Parser(Pos, _RR) then
              begin
                estado_atual := 4;
                dur_P      := AutomatoP.GetDur;
                Amp_P      := AutomatoP.GetAmp;
              end else
              begin
                for i := pos_ant - 1 to Pos do
                  primitivas.setWaveType(i,30);
                Pos := pos_ant;
              end;
             end else Pos := pos_ant;
        else
                atualize := True;
        end; // case
      end;
'e' : begin
        case estado_atual of
        0  :  if AutomatoQRS.Parser(Pos) then
              begin
                estado_atual := 6;
                dur_QRS      := AutomatoQRS.GetDur;
                Amp_QRS      := AutomatoQRS.GetAmp;
              end else
              begin
                for i := pos_ant - 1 to Pos do
                  primitivas.setWaveType(i,30);
                Pos := pos_ant;
              end;

        1  :  if AutomatoQRS.Parser(Pos) then
              begin
                estado_atual := 6;
                dur_QRS      := AutomatoQRS.GetDur;
                Amp_QRS      := AutomatoQRS.GetAmp;
              end else
              begin
                for i := pos_ant -1 to Pos do
                  primitivas.setWaveType(i,30);
                Pos := pos_ant;
              end;

        2  :  if AutomatoQRS.Parser(Pos) then
```

```
                begin
                  estado_atual := 6;
                  dur_QRS      := AutomatoQRS.GetDur;
                  Amp_QRS      := AutomatoQRS.GetAmp;
                end else
                begin
                  for i := pos_ant - 1 to Pos do
                    primitivas.setWaveType(i,30);
                  Pos := pos_ant;
                end;

          3  :  if AutomatoQRS.Parser(Pos) then
                begin
                  estado_atual := 6;
                  dur_QRS      := AutomatoQRS.GetDur;
                  Amp_QRS      := AutomatoQRS.GetAmp;
                end else
                begin
                  for i := pos_ant - 1 to Pos do
                    primitivas.setWaveType(i,30);
                  Pos := pos_ant;
                end;

          4  :  if AutomatoQRS.Parser(Pos) then
                begin
                  estado_atual := 6;
                  dur_QRS      := AutomatoQRS.GetDur;
                  Amp_QRS      := AutomatoQRS.GetAmp;
                end else
                begin
                  for i := pos_ant - 1 to Pos do
                    primitivas.setWaveType(i,30);
                  Pos := pos_ant;
                end;

          5  :  if AutomatoQRS.Parser(Pos) then
                begin
                  estado_atual := 6;
                  dur_QRS      := AutomatoQRS.GetDur;
                  Amp_QRS      := AutomatoQRS.GetAmp;
                end else
                begin
                  for i := pos_ant - 1 to Pos do
                    primitivas.setWaveType(i,30);
                  Pos := pos_ant;
                end;
          else
                  atualize := True;
          end; // case
        end;//begin
  end; // case
  if (((Pos  + 1) <= primitivas.getTopo - 1)and atualize) then
    Inc(Pos);
  primitivas.Get(Pos, car, Dur_i, Amp_i);
  tempo_atual := ((1/240*4)* (Pos-pos_def));
end; //while
dur_QRS := dur_QRS * _rate;
```

```
  dur_ST  := dur_ST  * _rate;
  dur_T   := dur_T   * _rate;
  dur_TP  := dur_TP  * _rate;
  dur_P   := dur_P   * _rate;
  dur_PQ  := dur_PQ  * _rate;

  if dur_QRS > _dur_QRS then
    _dur_QRS := dur_QRS;
  if dur_ST > _dur_ST then
    _dur_ST  := dur_ST;
  if dur_T > _dur_T then
    _dur_T   := dur_T;
  if dur_TP > _dur_TP then
    _dur_TP  := dur_TP;
  if dur_P > _dur_P then
    _dur_P   := dur_P;
  if dur_PQ > _dur_PQ then
    _dur_PQ  := dur_PQ;
  if Amp_QRS > _Amp_QRS then
    _Amp_QRS := Amp_QRS;
  if Amp_T > _Amp_T then
    _Amp_T   := Amp_T;
  if Amp_P > _Amp_P then
    _Amp_P   := Amp_P;
end;

end.
```

8.3.4 – Classe Tclassificador

```
unit uClassificador;
interface
uses uLista_primitivas, uAutomatoECG, Math, SysUtils;
type TClassificador = class
  protected
    ECG : TAutomatoECG;
    lista_aVF : TLista_DRG;
  public
    listas    : array[1..12] of TLista_DRG;
    numeroCanais : integer;
    rate : integer;
    tempoQrs, tempoT, tempoP, tempoST, tempoTP, tempoPQ,
    ampQrs, ampT, ampP, ampST, ampTP, ampPQ : double;

    constructor Create;
    destructor Destroy;
    procedure carregue_aVF(lista : TLista_DRG);
    procedure classifique();
    procedure parseAVF();
    procedure colete_amostras();
    procedure distribuaPadroes;
    function  LabelValue(x1,x2,y1,y2 : Double) : char;
    procedure inicialize;
end;
implementation
{ TClassificador }
```

```
procedure TClassificador.carregue_aVF(lista: TLista_DRG);
var i   : integer;
    amp : Double;
begin
  lista_aVF.initialize;
  for i := 0 to lista.getTopo - 1 do
  begin
    lista.Get( i, amp );
    lista_aVF.Add(amp);
  end;
end;

procedure TClassificador.classifique;
begin
  Self.colete_amostras;
  Self.parseAVF;
  Self.DistribuaPadroes;
end;

procedure TClassificador.colete_amostras;
var
  i, cont : integer;
  y, x_old, y_old : double;
  lab : string;
begin
  cont := 0;
  x_old := 0;
  y_old := 0;
  ECG.primitivas.initialize;
  for i := 0 to lista_aVF.getTopo do
  begin
    lista_aVF.Get(i, y);
    inc(cont);
    if cont = rate then
    begin
      lab := LabelValue(x_old,i,y_old,y);
      ECG.primitivas.Add(lab[1],1/(240), y - y_old,i,y) ;
      x_old := i;
      y_old := y;
      cont := 0;
    end;
  end;
end;

constructor TClassificador.Create;
begin
  ECG := TAutomatoECG.Create();
  lista_aVF := TLista_DRG.Create;
  inicialize;
end;

destructor TClassificador.Destroy;
begin
  ECG.Destroy;
end;
```

```
procedure TClassificador.distribuaPadroes;
var
  v_type, cont, i, j, k : Integer;
  x, y : Double;
begin
  cont := 0; j := 0;
  for i := 0 to lista_aVF.getTopo do
  begin
    v_type := ECG.primitivas.getWaveType(j,x,y);
    for k := 1 to numeroCanais  do
      listas[k].setWaveType(i,  v_type);
    inc(cont);
    if cont = rate then
    begin
      cont := 0;
      inc(j) ;
    end;
  end;
end;

procedure TClassificador.inicialize;
begin
  numeroCanais := 0;
  rate := 4;
  tempoQrs := 0;
  tempoT   := 0;
  tempoP   := 0;
  tempoST  := 0;
  tempoTP  := 0;
  tempoPQ  := 0;
  ampQrs   := 0;
  ampT     := 0;
  ampP     := 0;
  ampST    := 0;
  ampTP    := 0;
  ampPQ    := 0;
end;

function TClassificador.LabelValue(x1, x2, y1, y2: Double): char;
var
  fat, ang, tan : double;
  car : char;
begin
  // Alteração no intervalo do label 'a' - > [-10 .. 10]
  fat := 35;
  ang := arcTan2((y2 - y1),((x2 - x1)*fat));
  ang := RadToDeg(ang);
  if ((ang <= 90) and (ang > 75))then car := 'c'
  else
   if ((ang <= 75) and (ang > 10))then car := 'b' //>20
   else
    if ((ang <= 10) and (ang >-10))then car := 'a'  //<=20
    else
     if((ang <=-10) and (ang >-75))then car := 'd'
     else
      if((ang <= -75) and (ang >-90)) then car := 'e';
   result := car;
```

```
end;

procedure TClassificador.parseAVF;
var
  pos, _rate : Integer;
  vRR        : Double;
begin
  pos := 0;
  vRR := 0;
  _rate := rate;
  ECG.Initialize;
  ECG.returnTimeRR(Pos, _rate);
  vRR := ECG.GetRR;
  while pos < ECG.primitivas.getTopo - 1 do
  begin
    ECG.Initialize;
    ECG.Parser(pos, vRR, _rate);
  end;
  tempoQrs := ECG._dur_QRS;
  tempoT   := ECG._dur_T;
  tempoP   := ECG._dur_P;
  tempoST  := ECG._dur_ST;
  tempoTP  := ECG._dur_TP;
  tempoPQ  := ECG._dur_PQ;
  ampQrs   := ECG._Amp_QRS;
  ampT     := ECG._Amp_T;
  ampP     := ECG._Amp_P;
end;
end.
```

8.3.5 – Classe TAutomatoQRS

```
unit uAutomatoQRS;

interface
uses uLista_primitivas, uAutomato;
 type TAutomatoQRS = class (TAutomato)
   protected
     VEF: array [0..4] of integer;
     Estados: array [0..4] of integer;
     TabelaDeTransicao: array[0..4,1..5] of integer;
     function transicao(qi : integer; car : char) : integer;
   public
     constructor Create(_primitivas : TLista_primitivas);   override;
     procedure Initialize; override;
     function Parser( var Pos : integer) : Boolean; override;
end;

implementation
{ TAutomatoQRS }
constructor TAutomatoQRS.Create(_primitivas : TLista_primitivas);
var i : integer;
begin
  inherited Create(_primitivas);
  VEF[0] := (0);
  VEF[1] := (0);
```

```
  VEF[2] := (0);
  VEF[3] := (1);
  VEF[4] := (1);
  for i:= 0 to 4 do
    Estados[i]:= (i);
  (*Inicialização da tabela de simbolos*)
  (*  Estado 0 :  *)
  TabelaDeTransicao[0,1] := 4;     //(0,'a' )-> 4 Erro
  TabelaDeTransicao[0,2] := 2;     //(0,'c')-> 2
  TabelaDeTransicao[0,3] := 2;     //(0,'c')-> 2
  TabelaDeTransicao[0,4] := 1;     //(0,'c')-> 1
  TabelaDeTransicao[0,5] := 1;     //(0,'e')-> 1
  (*  Estado 1 :  *)
  TabelaDeTransicao[1,1] := 4;     //(1,'a' )-> 4
  TabelaDeTransicao[1,2] := 2;     //(1,'c')-> 2
  TabelaDeTransicao[1,3] := 2;     //(1,'c')-> 2
  TabelaDeTransicao[1,4] := 1;     //(1,'d')-> 1
  TabelaDeTransicao[1,5] := 1;     //(1,'e')-> 1
  (*  Estado 2 :  *)
  TabelaDeTransicao[2,1] := 3;
  TabelaDeTransicao[2,2] := 2;
  TabelaDeTransicao[2,3] := 2;
  TabelaDeTransicao[2,4] := 1;
  TabelaDeTransicao[2,5] := 1;
  (*  Estado 3 :  *)
  for i := 1 to 4 do
    TabelaDeTransicao[3,i] := 3;
  (*  Estado 4 :  *)
  for i := 1 to 4 do
    TabelaDeTransicao[4,i] := 4;
end;

procedure TAutomatoQRS.Initialize;
begin
  inherited;
end;

function TAutomatoQRS.Parser( var Pos : integer) : Boolean;
var
  car          : char;
  est          : integer;
  continuar    : boolean;
  Dur_i, Amp_i : Double;
begin
  continuar := True;
  primitivas.Get(pos,car,Dur_i,Amp_i);
  est:=0;
  Self.Dur := (1/(240));
  if Amp_i < 0 then
    Amp_i := -1*Amp_i;
  Amp := Amp_i;
  while continuar do
  begin
    est := transicao(est,car);
    if VEF[est] = 1 then
    begin
      continuar := False;
```

```
      if est = 4 then Dec(Pos);
    end else
    begin
      Dur := Dur + (1/240);
      if Amp_i < 0 then
        Amp_i := -1*Amp_i;
      Amp := Amp + Amp_i;
      primitivas.setWaveType(Pos,1);
      if pos + 1 < primitivas.getTopo then
      begin
        primitivas.Get(pos + 1,car,Dur_i,Amp_i);
        Inc(Pos);
      end else
        continuar := False;
    end;
  end; // while
  if est = 3 then
    result := True
  else
    result := False;
end;

function TAutomatoQRS.transicao(qi: integer; car: char): integer;
var
   i,indice:integer;
begin
  for i := 1 to 5 do
  begin
    if Simbolos[i] = car then
       indice:= i;
  end;
  result:= TabelaDeTransicao[qi,indice];
end;
end.
```

8.3.6 – Classe TAutomatoST

```
unit uAutomatoST;
interface
uses uLista_primitivas, uAutomato;
 type TAutomatoST = class (TAutomato)
   protected
     VEF: array [0..4] of integer;
     Estados: array [0..4] of integer;
     TabelaDeTransicao: array[0..4,1..5] of integer;
     function transicao(qi : integer; car : char) : integer;
   public
     constructor Create(_primitivas : TLista_primitivas);   override;
     procedure Initialize; override;
     function Parser(var Pos: integer): Boolean; override;
end;

implementation
{ TAutomatoST }
constructor TAutomatoST.Create(_primitivas : TLista_primitivas);
var i : integer;
```

```
begin
  inherited Create(_primitivas);
  VEF[0] := (0);
  VEF[1] := (0);
  VEF[2] := (0);
  VEF[3] := (1);
  VEF[4] := (1);
  for i:= 0 to 4 do
    Estados[i]:= (i);
  (*Inicialização da tabela de simbolos*)
  (*  Estado 0 :  *)
  TabelaDeTransicao[0,1] := 1;     //(0,'a')-> 1
  for i := 2 to 5 do
    TabelaDeTransicao[0,i] := 4;   //(0,<>'a')-> 2 Erro
  (*  Estado 1 :  *)
  TabelaDeTransicao[1,1] := 2;     //(1,'a')-> 1
  for i := 2 to 5 do
    TabelaDeTransicao[1,i] := 4;   //(1,<>'a')-> 2 Erro
  (*  Estado 2 :  *)
  TabelaDeTransicao[2,1] := 2;     //(2,'a')-> 2
  for i := 2 to 5 do
    TabelaDeTransicao[2,i] := 3;   //(2,<>'a')-> 3
  (*  Estado 3 :  *)
  for i := 1 to 5 do
    TabelaDeTransicao[3,i] := 3;   //(3,<>' ')-> 3
  (*  Estado 4 :  *)
  for i := 1 to 5 do
    TabelaDeTransicao[4,i] := 4;   //(4,<>' ')-> 4
end;

procedure TAutomatoST.Initialize;
begin
  inherited;
end;
function TAutomatoST.Parser(var Pos: integer): Boolean;
var
  car          : char;
  est     : integer;
  continuar    : boolean;
  Dur_i, Amp_i : Double;
begin
  continuar := True;
  primitivas.Get(pos,car,Dur_i,Amp_i);
  Self.Dur := (1/(240));
  Amp := Amp_i;
  est:=0;
  while continuar do
  begin
    primitivas.setWaveType(Pos,2);
    est := transicao(est,car);
    if VEF[est] = 1 then
    begin
      continuar := False;
      if est = 4 then pos := pos - 1;
    end else
    begin
      if pos < primitivas.getTopo then
```

```
      begin
        Inc(pos);
        primitivas.Get(pos,car,Dur_i,Amp_i);
        Dur := Dur + (1/240);
        Amp := Amp + Amp_i;
      end else
        continuar := False;
    end;
  end; // while
  if est = 3 then
     result := True
  else
     result := False;
end;


function TAutomatoST.transicao(qi: integer; car: char): integer;
var
   i,indice:integer;
begin
  for i := 1 to 5 do
  begin
    if Simbolos[i] = car then
       indice:= i;
  end;
  result:= TabelaDeTransicao[qi,indice];
end;
end.
```

8.3.7 – Classe TAutomatoT

```
unit uAutomatoT;
interface
uses uLista_primitivas, uAutomato;
 type TAutomatoT = class (TAutomato)
   protected
     VEF: array [0..8] of integer;
     Estados: array [0..8] of integer;
     TabelaDeTransicao: array[0..8,1..5] of integer;
     function transicao(qi : integer; car : char) : integer;
   public
     constructor Create(_primitivas : TLista_primitivas);   override;
     procedure Initialize; override;
     function Parser( var Pos : integer; _RR : Double ) : Boolean;
     function avaliaParametros(_Amp_l, _Amp_r, _Dur, _Dur_t, RR : Double)
: Boolean;
end;

implementation
{ TAutomatoT }
function TAutomatoT.avaliaParametros(_Amp_l, _Amp_r, _Dur, _Dur_t,
  RR: Double): Boolean;
begin
  if(   ( _Amp_l  > (0.5 * _Amp_r))
    and ( _Amp_l  < (1.5 * _Amp_r))
    and ((_Dur/2) > _Dur_t        )
```

```
    and ( _Dur     > (0.1 * RR)    )
    and ( _Dur     < (0.4 * RR)    ) ) then
      result := True
  else
      result := False;
end;

constructor TAutomatoT.Create(_primitivas : TLista_primitivas);
var i : integer;
begin
  inherited Create(_primitivas);
  VEF[0] := (0);
  VEF[1] := (0);
  VEF[2] := (0);
  VEF[3] := (0);
  VEF[4] := (0);
  VEF[5] := (0);
  VEF[6] := (0);
  VEF[7] := (1);
  VEF[8] := (1);
  for i:= 0 to 8 do
    Estados[i]:= (i);
  (*Inicialização da tabela de simbolos*)
  (*  Estado 0 :  *)
  for i := 1 to 5 do
    TabelaDeTransicao[0,i] := 8;   //(0,'a'|'c'|'e' )-> 8 Erro
  TabelaDeTransicao[0,2] := 1;     //(0,'b')-> 1
  TabelaDeTransicao[0,4] := 2;     //(0,'d')-> 4
  (*  Estado 1 :  *)
  TabelaDeTransicao[1,1] := 2;     //(1,'a')-> 2
  TabelaDeTransicao[1,2] := 1;     //(1,'b')-> 1
  TabelaDeTransicao[1,3] := 8;     //(1,'c')-> 8  Erro
  TabelaDeTransicao[1,4] := 3;     //(1,'d')-> 3
  TabelaDeTransicao[1,5] := 3;     //(1,'e')-> 3
  (*  Estado 2 :  *)
  TabelaDeTransicao[2,1] := 2;     //(2,'a')-> 2
  TabelaDeTransicao[2,2] := 8;     //(2,'b')-> 8  Erro
  TabelaDeTransicao[2,3] := 8;     //(2,'c')-> 8  Erro
  TabelaDeTransicao[2,4] := 3;     //(2,'d')-> 3
  TabelaDeTransicao[2,5] := 3;     //(2,'e')-> 3
  (*  Estado 3 :  *)
  TabelaDeTransicao[3,1] := 7;     //(3,'a')-> 7
  TabelaDeTransicao[3,2] := 7;     //(3,'b')-> 7
  TabelaDeTransicao[3,3] := 7;     //(3,'c')-> 7
  TabelaDeTransicao[3,4] := 3;     //(3,'d')-> 3
  TabelaDeTransicao[3,5] := 3;     //(3,'e')-> 3
  (*  Estado 4 :  *)
  TabelaDeTransicao[4,1] := 5;     //(4,'a')-> 5
  TabelaDeTransicao[4,2] := 6;     //(4,'b')-> 6
  TabelaDeTransicao[4,3] := 6;     //(4,'c')-> 6
  TabelaDeTransicao[4,4] := 4;     //(4,'d')-> 4
  TabelaDeTransicao[4,5] := 8;     //(4,'e')-> 8  Erro
  (*  Estado 5 :  *)
  TabelaDeTransicao[5,1] := 5;     //(5,'a')-> 5
  TabelaDeTransicao[5,2] := 6;     //(5,'b')-> 6
  TabelaDeTransicao[5,3] := 6;     //(5,'c')-> 6
  TabelaDeTransicao[5,4] := 8;     //(5,'d')-> 8  Erro
```

```
  TabelaDeTransicao[5,5] := 8;     //(5,'e')-> 8  Erro
  (*  Estado 6 :  *)
  TabelaDeTransicao[6,1] := 7;     //(6,'a')-> 7
  TabelaDeTransicao[6,2] := 6;     //(6,'b')-> 6
  TabelaDeTransicao[6,3] := 6;     //(6,'c')-> 6
  TabelaDeTransicao[6,4] := 7;     //(6,'d')-> 7
  TabelaDeTransicao[6,5] := 7;     //(6,'e')-> 7
  (*  Estado 7 :  *)
  for i := 1 to 5 do
    TabelaDeTransicao[7,i] := 7;
  (*  Estado 8 :  *)
  for i := 1 to 5 do
    TabelaDeTransicao[8,i] := 8;
end;

procedure TAutomatoT.Initialize;
begin
  inherited;
end;

function TAutomatoT.Parser(var Pos: integer; _RR: Double): Boolean;
var
  car          : char;
  est          : integer;
  continuar    : boolean;
  Dur_i, Dur_T,
  Amp_i, Amp_r,
  Amp_l        : Double;
begin
  continuar := True;
  primitivas.Get(pos,car,Dur_i,Amp_i);
  Self.Dur := (1/(240));
  Self.Amp := 0;
  if Amp_i < 0 then
    Amp_i := -1*Amp_i;
  Amp_l := Amp_i;
  Dur_T := 0;
  Amp_r := 0;
  est   := 0;
  while continuar do
  begin
    primitivas.setWaveType(Pos,3);
    est := transicao(est,car);
    if  (avaliaParametros(Amp_l,Amp_r, Dur, Dur_T, _RR))
    or  (  VEF[est] = 1 )                                then
      continuar := False
    else
    begin
      if pos < primitivas.getTopo then
      begin
        Inc(pos);
        primitivas.Get(pos,car,Dur_i,Amp_i);
        Dur := Dur + (1/(240));
        if Amp_i < 0 then
          Amp_i := -1*Amp_i;
        if (est = 1)or(est = 4)then // subindo...
           Amp_l := Amp_l + Amp_i
```

```
        else
        if (est = 2)or(est = 5)then // plato...
          Dur_T := Dur_T + (1/(240))
        else
        if (est = 3)or(est = 6)then // descendo...
          Amp_r := Amp_r + Amp_i;
      end else
        continuar := False;
    end;
  end; // while
  Amp := (Amp_l + Amp_r)/2;
  if est <> 8 then
  begin
    result := True;
  end else
    result := False;
end;


function TAutomatoT.transicao(qi: integer; car: char): integer;
var
   i,indice:integer;
begin
  for i := 1 to 5 do
  begin
    if Simbolos[i] = car then
       indice:= i;
  end;
  result:= TabelaDeTransicao[qi,indice];
end;
end.
```

8.3.8 – Classe TAutomatoTP

```
unit uAutomatoTP;
interface
uses uLista_primitivas, uAutomato;
 type TAutomatoTP = class (TAutomato)
   protected
     VEF: array [0..2] of integer;
     Estados: array [0..2] of integer;
     TabelaDeTransicao: array[0..2,1..5] of integer;
     function transicao(qi : integer; car : char) : integer;
   public
     constructor Create(_primitivas : TLista_primitivas);   override;
     procedure Initialize; override;
     function Parser( var Pos : integer) : Boolean;     override;
 end;

implementation
{ TAutomatoTP }
constructor TAutomatoTP.Create(_primitivas : TLista_primitivas);
var i : integer;
begin
  inherited Create(_primitivas);
  VEF[0] := (0);
```

```
  VEF[1] := (0);
  VEF[2] := (1);
  for i:= 0 to 2 do
    Estados[i]:= (i);
  (*Inicialização da tabela de simbolos*)
  (*  Estado 0 :  *)
  TabelaDeTransicao[0,1] := 1;     //(0,'a')-> 1
  for i := 2 to 5 do
    TabelaDeTransicao[0,i] := 2;   //(0,<>'a')-> 2 Erro
  (*  Estado 1 :  *)
  TabelaDeTransicao[1,1] := 1;     //(1,'a')-> 1
  for i := 2 to 5 do
    TabelaDeTransicao[1,i] := 2;   //(1,<>'a')-> 2 Erro
  (*  Estado 2 :  *)
  for i := 1 to 5 do
    TabelaDeTransicao[2,i] := 2;   //(2,'..')-> 2 Erro
end;

procedure TAutomatoTP.Initialize;
begin
  inherited Initialize;
end;

function TAutomatoTP.Parser(var Pos: integer): Boolean;
var
  car          : char;
  est     : integer;
  continuar    : boolean;
  Dur_i, Amp_i : Double;
begin
  continuar := True;
  primitivas.Get(pos,car,Dur_i,Amp_i);
  Self.Dur := (1/(240));
  Amp := Amp_i;
  est:=0;
  while continuar do
  begin
    primitivas.setWaveType(Pos,4);
    est := transicao(est,car);
    if VEF[est] = 1 then
    begin
      continuar := False;
    end else
    begin
      if pos < primitivas.getTopo then
      begin
        Inc(pos);
        primitivas.Get(pos,car,Dur_i,Amp_i);
        Dur := Dur + (1/240);
        Amp := Amp + Amp_i;
      end else
        continuar := False;
    end;
  end; // while
  result := True;
end;
```

```
function TAutomatoTP.transicao(qi: integer; car: char): integer;
var
   i,indice:integer;
begin
  for i := 1 to 5 do
  begin
    if Simbolos[i] = car then
       indice:= i;
  end;
  result:= TabelaDeTransicao[qi,indice];
end;
end.
```

8.3.9 – Classe TAutomatoP

```
unit uAutomatoP;
interface
uses uLista_primitivas, uAutomato;
 type TAutomatoP = class (TAutomato)
   protected
     VEF: array [0..8] of integer;
     Estados: array [0..8] of integer;
     TabelaDeTransicao: array[0..8,1..5] of integer;
     function transicao(qi : integer; car : char) : integer;
   public
     constructor Create(_primitivas : TLista_primitivas);   override;
     procedure Initialize; override;
     function Parser( var Pos : integer; _RR : Double) : Boolean;
     function avaliaParametros(_Dur, RR : Double) : Boolean;
end;
implementation
{ TAutomatoP }
constructor TAutomatoP.Create(_primitivas : TLista_primitivas);
var i : integer;
begin
  inherited Create(_primitivas);
  VEF[0] := (0);
  VEF[1] := (0);
  VEF[2] := (0);
  VEF[3] := (0);
  VEF[4] := (0);
  VEF[5] := (0);
  VEF[6] := (0);
  VEF[7] := (1);
  VEF[8] := (1);
  for i:= 0 to 8 do
    Estados[i]:= (i);
  (*Inicialização da tabela de simbolos*)
  (*  Estado 0 :  *)
  for i := 1 to 5 do
    TabelaDeTransicao[0,i] := 8;   //(0,'a'|'c'|'e' )-> 8 Erro
  TabelaDeTransicao[0,2] := 1;     //(0,'b')-> 1
  TabelaDeTransicao[0,4] := 4;     //(0,'d')-> 4
   (*  Estado 1 :  *)
  TabelaDeTransicao[1,1] := 2;     //(1,'a')-> 2
  TabelaDeTransicao[1,2] := 1;     //(1,'b')-> 1
  TabelaDeTransicao[1,3] := 8;     //(1,'c')-> 8  Erro
  TabelaDeTransicao[1,4] := 3;     //(1,'d')-> 3
  TabelaDeTransicao[1,5] := 3;     //(1,'e')-> 3
  (*  Estado 2 :  *)
  TabelaDeTransicao[2,1] := 2;     //(2,'a')-> 2
  TabelaDeTransicao[2,2] := 8;     //(2,'b')-> 8  Erro
  TabelaDeTransicao[2,3] := 8;     //(2,'c')-> 8  Erro
  TabelaDeTransicao[2,4] := 3;     //(2,'d')-> 3
  TabelaDeTransicao[2,5] := 3;     //(2,'e')-> 3
   (*  Estado 3 :  *)
  TabelaDeTransicao[3,1] := 7;     //(3,'a')-> 7
  TabelaDeTransicao[3,2] := 7;     //(3,'b')-> 7
```

```
  TabelaDeTransicao[3,3] := 7;     //(3,'c')-> 7
  TabelaDeTransicao[3,4] := 3;     //(3,'d')-> 3
  TabelaDeTransicao[3,5] := 3;     //(3,'e')-> 3
  (*  Estado 4 :  *)
  TabelaDeTransicao[4,1] := 5;     //(4,'a')-> 5
  TabelaDeTransicao[4,2] := 6;     //(4,'b')-> 6
  TabelaDeTransicao[4,3] := 6;     //(4,'c')-> 6
  TabelaDeTransicao[4,4] := 4;     //(4,'d')-> 4
  TabelaDeTransicao[4,5] := 8;     //(4,'e')-> 8  Erro
  (*  Estado 5 :  *)
  TabelaDeTransicao[5,1] := 5;     //(5,'a')-> 5
  TabelaDeTransicao[5,2] := 6;     //(5,'b')-> 6
  TabelaDeTransicao[5,3] := 6;     //(5,'c')-> 6
  TabelaDeTransicao[5,4] := 8;     //(5,'d')-> 8  Erro
  TabelaDeTransicao[5,5] := 8;     //(5,'e')-> 8  Erro
  (*  Estado 6 :  *)
  TabelaDeTransicao[6,1] := 7;     //(6,'a')-> 7
  TabelaDeTransicao[6,2] := 6;     //(6,'b')-> 6
  TabelaDeTransicao[6,3] := 6;     //(6,'c')-> 6
  TabelaDeTransicao[6,4] := 7;     //(6,'d')-> 7
  TabelaDeTransicao[6,5] := 7;     //(6,'e')-> 7
 (*  Estado 7 :  *)
  for i := 1 to 5 do
    TabelaDeTransicao[7,i] := 7;
  (*  Estado 8 :  *)
  for i := 1 to 5 do
    TabelaDeTransicao[8,i] := 8;
end;

procedure TAutomatoP.Initialize;
begin
  inherited;
end;

function TAutomatoP.transicao(qi: integer; car: char): integer;
var
   i,indice:integer;
begin
  for i := 1 to 5 do
  begin
    if Simbolos[i] = car then
       indice:= i;
  end;
  result:= TabelaDeTransicao[qi,indice];
end;

function TAutomatoP.Parser(var Pos: integer; _RR : Double): Boolean;
var
  car          : char;
  est          : integer;
  continuar    : boolean;
  Dur_i, Amp_i : Double;
begin
  continuar := True;
  primitivas.Get(pos,car,Dur_i,Amp_i);
  est := 0;
  Self.Dur := (1/(240));
```

```
  if Amp_i < 0 then
    Amp_i := -1*Amp_i;
  self.Amp := Amp_i;
  while continuar do
  begin
    primitivas.setWaveType(Pos,5);
    est := transicao(est,car);
    if  (avaliaParametros(Dur, _RR))
    or  (  VEF[est] = 1 )            then
      continuar := False
    else
    begin
      if pos < primitivas.getTopo then
      begin
        Inc(pos);
        primitivas.Get(pos,car,Dur_i,Amp_i);
        Dur := Dur + (1/(240));
        if Amp_i < 0 then
         Amp_i := -1*Amp_i;
        Amp := Amp + Amp_i;
      end else
        continuar := False;
    end;
  end; // while
  if est <> 8 then
    result := True
  else
    result := False;
end;

function TAutomatoP.avaliaParametros(_Dur, RR : Double) : Boolean;
begin
  if    ( _Dur     > (0.1 * RR)    )
    and ( _Dur     < (0.11 * RR)    )  then
      result := True
  else
      result := False;
end;
end.
```

8.3.10 – Classe TAutomatoPQ

```
unit uAutomatoPQ;
interface
uses uLista_primitivas, uAutomato;
 type TAutomatoPQ = class (TAutomato)
   protected
     VEF: array [0..2] of integer;
     Estados: array [0..2] of integer;
     TabelaDeTransicao: array[0..2,1..5] of integer;
     function transicao(qi : integer; car : char) : integer;
   public
     constructor Create(_primitivas : TLista_primitivas);   override;
     procedure Initialize; override;
     function Parser( var Pos : integer) : Boolean;     override;
end;
implementation
{ TAutomatoPQ }
constructor TAutomatoPQ.Create(_primitivas : TLista_primitivas);
var i:integer;
begin
  inherited Create(_primitivas);
  VEF[0] := (0);
  VEF[1] := (0);
  VEF[2] := (1);
  for i:= 0 to 2 do
    Estados[i]:= (i);
  (*Inicialização da tabela de simbolos*)
  (*  Estado 0 :  *)
  TabelaDeTransicao[0,1] := 1;  //(0,'a')-> 1
  for i := 2 to 5 do
    TabelaDeTransicao[0,i] := 2;  //(0,<>'a')-> 2 Erro
  (*  Estado 1 :  *)
  TabelaDeTransicao[1,1] := 1;  //(1,'a')-> 1
  for i := 2 to 5 do
    TabelaDeTransicao[1,i] := 2;  //(1,<>'a')-> 2 Erro
  (*  Estado 2 :  *)
  for i := 1 to 5 do
    TabelaDeTransicao[2,i] := 2;  //(2,'..')-> 2 Erro
  inherited;
end;

procedure TAutomatoPQ.Initialize;
begin
  inherited;
end;

function TAutomatoPQ.Parser( var Pos : integer) : Boolean;
var
  car          : char;
  est          : integer;
  continuar    : boolean;
  Dur_i, Amp_i : Double;
begin
  continuar := True;
```

```
  primitivas.Get(pos,car,Dur_i,Amp_i);
  Self.Dur := (1/(240));
  Amp := Amp_i;
  est:=0;
  while continuar do
  begin
    primitivas.setWaveType(Pos,6);
    est := transicao(est,car);
    if VEF[est] = 1 then
    begin
      continuar := False;
      Dec(pos);
    end else
    begin
      if pos < primitivas.getTopo then
      begin
        Inc(pos);
        primitivas.Get(pos,car,Dur_i,Amp_i);
        Dur := Dur + (1/240);
        Amp := Amp + Amp_i;
      end else
        continuar := False;
    end;
  end; // while
  result := True;
end;

function TAutomatoPQ.transicao(qi: integer; car: char): integer;
var
   i,indice:integer;
begin
  for i := 1 to 5 do
  begin
    if Simbolos[i] = car then
       indice:= i;
  end;
  result:= TabelaDeTransicao[qi,indice];
end;
end.
```

9. REFERÊNCIAS BIBLIOGRÁFICAS

[1] V. Pilla, H. S. Lopes. **Reconhecimento de Padrões em Sinais Eletrocardiográficos com Rede Neurofuzzy e Algoritmos Genéticos**, IV Congresso Brasileiro de Redes Neurais pp. 042-046, July 20-22, 1999 - ITA, São José dos Campos - SP – Brazil

[2] A. Koski, M. Juhola, M. Meriste. **Syntactic Recognition of ECG Signals by Attributed Finite Automata,** Pattern Recognition, Vol 28, No12, pp 1927-1940 Pattern Recognition Society – Great Britain , 1995

[3] J. Bardonova, I. Provazník. **Hidden Markov Models in Wavelet Analysis,** Programme of Brno University of Technology, 1998.

[4] M. Ohlsson, H. Host, L. Edenbrandt. **Acute Myocardial Infaction: Analysis of the ECG Using Artificial Neural Networks,** Lund University, 1999.

[5] D. Frenkel, J. Nadal, **Comparação de Métodos de representação do segmento ST na detecção automática de Isquemias Miocádicas,** Revista Brasileira de Engenharia Biomédica, v. 16, n. 3, p. 153-162, set/dez, 2000.

[6] F. L. Baldissera, A. Orth, M. R. Stemmer, **Análise da Transformada de Wavelets Aplicada ao Processamento Freqüencial de Sinais,** Simpósio Catarinense de Processamento Digital de Imagens, SCPDI, 2001.

[7] A. Koski, **Modelling ECG signals with hidden Markov models,** Elsevier Artificial Intelligente in Medicine, 8 453-471, 1996.

[8] S. S. COSTA, **Modelagem e Implementação Orientada a Objetos de um Cliente de Rede para Banco de Dados de Imagens Médicas Digitais utilizando o Padrão DICOM 3.0,** Universidade Federal de Santa Catarina, Florianópolis, dezembro de 1999.

[9] National Electrical Manufactures Association**,** Website**, Digital Imaging and Communications in Medicine (DICOM),** Parts 1-3**",** Virginia, 2000.

[10] AHO, A. V., ULLMAN J. D., SETHI, Ravi. **Compiladores – Princípios, Técnicas e Ferramentas**. Livros técnicos e científicos S.A., Rio de Janeiro, 1995.

[11] www1.terravista.pt/Enseada/8146/ECG.html

[12] www.medial.com.br/noticia.asp?cod=240

[13] www.cardiol.br

[14] www.ccs.uel.br/pbl/cardio/capitulo4.asp