Learning ECG Grammars

Gabriella Kókai[1], Zoltán Alexin[2], and Tibor Gyimóthy[3]

[1] Institute of Informatics, József Attila University
Árpád tér 2, H-6720 Szeged, Hungary
e-mail: kokai@inf.u-szeged.hu

[2] Department of Applied Informatics, József Attila University
Árpád tér 2, H-6720 Szeged, Hungary
e-mail: alexin@inf.u-szeged.hu

[3] Research Group on Artificial Intelligence
Hungarian Academy of Sciences,
Aradi vértanuk tere 1, H-6720 Szeged, Hungary
e-mail: gyimi@inf.u-szeged.hu.

Abstract. In this paper a learning system is presented which integrates an ECG waveform classifier (called PECG) with an interactive learner (called IMPUT). The PECG system is based on an attribute grammar specification of ECGs that has been transformed to Prolog. The IMPUT system combines the interactive debugging technique IDT with the unfolding algorithm introduced in SPECTRE. Using the IMPUT system we can effectively assist in preparing the correct description of the basic structures of ECG waveforms. The application of the system for learning ECG waveforms is demonstrated with the help of an example. [4]
Keywords: inductive logic programming, program specialization, syntactic pattern recognition

1 Introduction

In this paper a complex system is presented that is able to classify ECG waveforms described as a combination of primitives. The system helps the user to improve the classifier program if it cannot recognize correctly the given input waveforms.

The system integrates an ECG waveform classifier (called PECG) with an interactive learner (called IMPUT).

The PECG system is based on an attribute grammar specification of ECG waveforms published by Skordalakis et al [16, 17]. The basic idea of PECG, introduced in [8], is to integrate the ECG classifier program (implemented in Prolog) with the Interactive Diagnosis and Testing (IDT) module [14] and a

[4] This work was supported by the grant OTKA T14228 and the project PHARE TDQM 9305-02/1022 ("ILP2/HUN").
A short version of this paper was published in the proceedings of the AIME97 conference.

graphic viewer. This integrated tool can recognize whether any modification in the classifier program is needed. If the system cannot analyze the input then the user is helped by the built-in IDT debugger to find the false clause.

The learning part of the new system is based on an Inductive Logic Programming (ILP) [10, 12] method (called IMPUT) introduced in [1]. The IMPUT system combines the interactive debugging technique IDT with the specialization algorithm SPECTRE presented in [3].

In the PECG system the ECG waveforms are described in Definite Clause Grammars (DCGs). The construction of these grammars for different waveforms is a very sophisticated task, therefore a tool which can assist in this work would be very useful. That was the reason for us to develop an extended version of the PECG system.

The user of this system has to prepare an *over-generalized* grammar in the first step. This grammar may accept not only correct but many incorrect waveforms as well. Of course it is much more easier to prepare such *over-generalized* initial grammar than the correct one. The inputs of the extended PECG system are this initial grammar, some background knowledge, positive and negative examples of ECG waveforms to be inferred. The system uses an ILP learning method to improve the inital grammar such that the modified grammar accepts each positive example but no negative ones.

During the learning process the system asks the user about the correctness of the recognized subpatterns (subgrammars). To answer there questions may be very difficult, hence the system has been augmented with a graphic viewer module, which displays the part of the ECG waveform being currently analyzed. On the basis of the graphic display a medical expert can easily decide whether a subpattern of a given waveform has been correctly recognized or not. If an incorrect part of a grammar is identified then the system tries to apply an unfolding step to transform the grammar. This transformation usually results in a more precise recognizer of a given waveform. The system applies these transformation steps until the modified grammar does not recognize any negative example. By using this method correct recognizer grammars can be inferred for ECG waveforms starting from easy prepared *over-generalized* initial specification.

In Section 2 we briefly describe the recognition of ECG waveforms and its linguistic representation used by our system. Sections 3 contains short descriptions of the PECG, the IMPUT and the integrated system. In Section 4 the working of the integrated system is demonstrated. Finally, in Section 5 a brief summary and comments on possible future studies are presented.

2 A grammatical approach for the analysis of ECG waveforms

2.1 Recognition of ECG waveforms

In this section we give a short overview on the grammatical description of the ECG waveforms. The methodologies and notations are borrowed from the Skordalakis' paper [16].

The electrocardiograms are largely employed as a diagnostic tool in clinical practice. With the help of such diagnostic information certain heart abnormalities can be recognized. There are many ECG processing systems which use computers to recognize abnormal waveforms and to automating this process [2], [9], [16], [19].

In Figure 1. three complexes (P, QRS, T) are presented which can be observed as an ECG waveform. Every cardiac complex contains one or more cardiac waves, but the most important phase in the heart action is the QRS complex. We chose this part to demonstrate the work of our system.

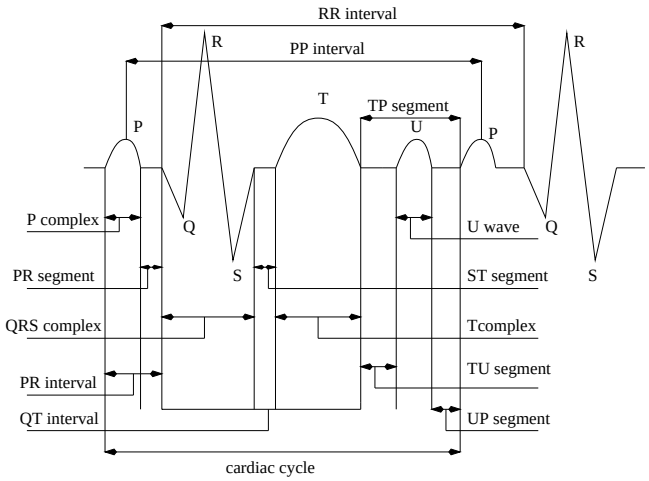

Fig. 1. The components of an ECG waveform

(*this figure is presented in* [16])

The basic parts of the ECG's are the waves. A segment can build up from waves. An interval contains one or more segments. The set of intervals gives us the complex, and a cardiac cycle can be decomposed on intervals. For example we can compose the Q, R, and S waves into an QRS complex. The other patterns in a cardiac cycle are the interwave segments (PR, ST, TU, UP, TP) and the intervals (PR, QT, RR, PP) as can be seen in Figure 1.

In the syntactic recognition of ECG there are four subproblems:

(1) primitive pattern selection
(2) primitive pattern extraction
(3) primitive pattern encoding
(4) formulation of a pattern grammar

For the first three subproblems details can be found in [16]. In this paper we will concentrate on the analysis problem of *pattern grammar*. We use three primitive

patterns: "peak", "straight line segment", and "parabolic segment". In the electrocardiographic pattern we can separate two main shapes; the complex and the segment subpatterns. A complex builds up from series of peaks. The segment can be divided into two other classes which are the straight line and the parabola.

2.2 The syntax of ECG grammar

In [16] an attribute grammar was used for the specification of ECG waveforms. An ECG waveform is described as a string of symbols from the alphabet $\Sigma = \{K^+, K^-, E, \Pi\}$ where K^+ and K^- represent the positive and negative peaks, E the straight line segment and Π the parabolic segment.

Our approach starts from attribute grammar specifications of waveforms, however we use Definite Clause Grammar (DCG) representations for the recognition of these waveforms. The system was extended with a graphic viewer to trace the history of the traversing of the proof tree of the analyzer program.

3 The architecture of the learning system

In this section first, we give a short overview of the PECG, and IMPUT systems. A more detailed description from these two systems can be found in [8], [1]. The remained part of this section deals with the description of the integrated tool.

3.1 PECG: A classifier program for ECG waveforms

The structure of the system can be seen in Figure 2.

The main parts are the following:

- *the ECG processing module:* this part of the system uses a syntactic approach to recognize an electrocardiographic pattern.
- *the IDT module:* this part is an interactive debugging and testing tool that is based on the Shapiro's Interactive Diagnosis Algorithm [15] and Category Partition Testing Method [13].
- *the graphic viewer module:* this part makes visible the current subpatterns being analyzed.

The ECG processing module contains a syntactical approach (introduced in [16]) for the recognition of ECG waveforms. In the *primitive pattern selection* part of this module an ECG waveform is decomposed into primitive patterns (straight line, peak, parabola). The main task of the *primitive pattern extraction* is to select the peaks. This part is based on recognition of the noisy peaks. In the *linguistic representation* part the result of the last two steps are represented as a string of symbols from an alphabet. As an example let us suppose that we have the following initial ECG waveform presented in Figure 3. The linguistic representation of it is the following:

$$EK^+\Pi K^+K^-EK^+\Pi K^+EK^+K^-\Pi K^+\Pi K^+\Pi K^+K^-EK^+\Pi$$

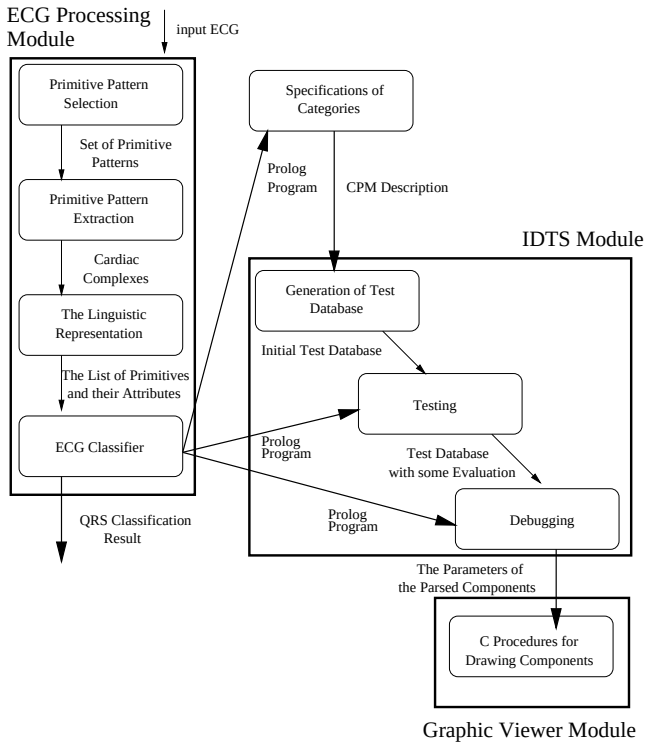

Fig. 2. The structure of the PECG system

Fig. 3. An initial ECG waveform

In Skordalakis' approach [16] many numeric values are associated with the primitives (e.g. boundaries of peaks, features of slopes, starting and ending points of segments). The PECG system uses these numeric features for the correct recognition of ECG waveforms. However, in this paper we concentrate only on the learning of the syntactical part, therefore these numeric features are not discussed here.

The *ECG classifier* module recognizes the ECG waveforms from their linguistic representations. The syntactical part of the classifier program (originally an attribute grammar rewritten in logical form) can be seen in Appendix 6.1.

The IDT method used in the next part of the PECG system is based on the algorithmic program debugging technique introduced by Shapiro [15]. This method can isolate an erroneous procedure, given a program and an input on which it behaves incorrectly. A major drawback of this debugging technique is the great number of queries made to the user about the correctness of intermediate results of procedure calls. An improvement in the bug localization process is realized in IDT by combining the *category partition testing method* [6], with the algorithm introduced in [15]. The main idea basically as follows: If the program has already been tested, the test results for the procedures of the program can be directly applied in the debugging process without consulting the user. The IDT interactive debugging environment was presented in [7, 14].

The IDT part can be divided into three submodules. The *test database* module generates the initial test database from a CPM specification. In the *testing* part a predicate can be chosen and the user can give a representing element of the testframe generated for this predicate. Then he or she can decide whether, for the given input, the output is correct or not. The *debugging* part is based on the idea that if the program has already been tested then the test results can be directly applied without asking the user *difficult* questions.

In the PECG system the algorithmic debugger IDT has been extended with a graphic viewer. This graphic module shows the ECG subpattern being analyzed and in this way provides effective assistance for the user to answer the questions invoked by the debugger. In Figure 4. the complete ECG waveform is displayed. In Figure 5. it can be seen, how the graphic viewer draws out the recognized

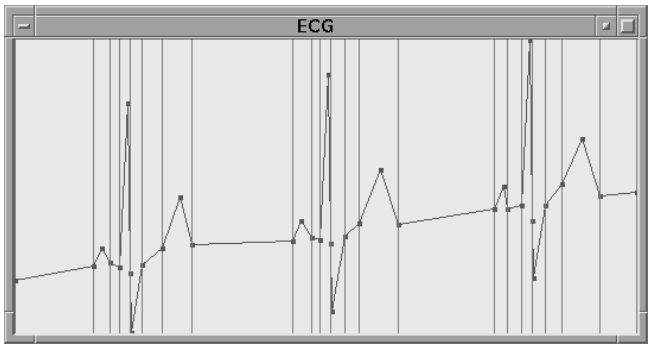

Fig. 4. The representation of the input ECG

QRS waveform in a different graphic style.

3.2 IMPUT: The learning system

In this section the IMPUT system is briefly presented. It consists of two main parts. The specialization algorithm comes from the SPECTRE system while the

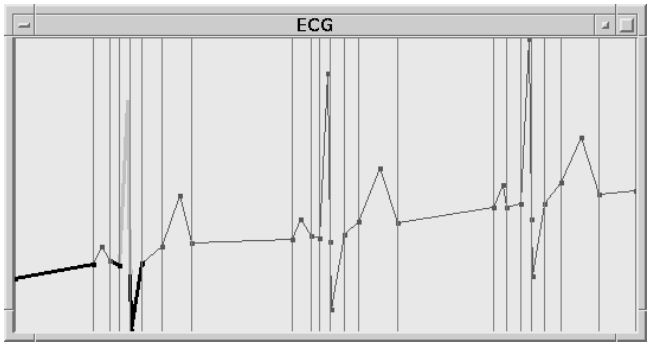

Fig. 5. The graphical representation of the recognized QRS part

Input: An initial program $\mathbf{P} = \{p_1, \ldots, p_u\}$, background knowledge $\mathbf{B} = \{b_1, \ldots, b_v\}$, the positive and the negative examples $\mathbf{E}^+$, $\mathbf{E}^-$.
Output: Series of programs $\mathbf{P}'^{(0)}$, $\mathbf{P}'^{(1)}$, ..., $\mathbf{P}'^{(n)}$ ($\mathbf{P}'^{(0)} = \mathbf{P}$), where $\mathbf{P}'^{(i+1)} = \tilde{\mathcal{U}}\left(\mathbf{P}'^{(i)}\right)$ $(0 \leq i < n)$, $\tilde{\mathcal{U}}$ is the unfolding operator extended with clause removal.

The Algorithm:

Check if the program $\mathbf{P}$ terminates on all $e^+ \in \mathbf{E}^+$.
if fails then stop "Initial program should cover all positive examples."
let $\mathbf{i} = 0$.
while there is an $e^- \in \mathbf{E}^-$ such that $\mathbf{P}'^{(i)}$ does not fail on e^- do
begin
  Find a buggy clause $c \in \mathbf{P}'^{(i)}$ using the IDT debugger
  Perform unfolding on c using a computation rule.
  let $\mathbf{C} = \{c_1, \ldots, c_s\}$ be the resolvents of c.
  Remove from $\mathbf{C}$ all those clauses that does not occur in
    refutations of positive examples.
  let $\mathbf{P}'^{(i+1)} = \mathbf{P}'^{(i)} \setminus \{c\} \cup \mathbf{C}$
  let $\mathbf{i} = \mathbf{i} + 1$
end

Fig. 6. The basic algorithm of IMPUT

interactive debugger part is imported from the IDT. The original specialization algorithm was extended in order to be able to revise multiple predicates simultaneously, to be able to read in DCG rules and to handle recursive predicates.

The algorithm SPECTRE [3] specializes logic programs with respect to positive and negative examples by applying the transformation rule *unfolding* [18] together with clause removal. The choice of which literal to unfold upon is made using a computation rule, which is given as input to the algorithm. A detailed description of SPECTRE can be found in [3].

The IMPUT system integrates the SPECTRE and the IDT methods. The main idea of IMPUT is that the identification of the next clause to be unfolded has a crucial importance in the effectiveness of the specialization process. The specialization steps are invoked by the negative examples that are covered by the target predicate. We assume that when a negative example is covered by the current version of the program, then there is at least one clause which is responsible for this incorrect covering. IMPUT uses the IDT debugging algorithm to identify a buggy clause of the program. The clause identified in this process will be unfolded in the next step of the specialization algorithm. The basic algorithm of IMPUT system is presented in Figure 6.

3.3 The extended PECG system

The extended PECG system involves an integration of PECG with the IMPUT interactive learning system. The inputs of this system are an *over-generalized* grammar for the ECG waveforms, the background knowledge, positive and negative examples of the ECG waveforms to be inferred. The system uses the IMPUT method to improve the initial *over-generalized* grammar such that the modified grammar accepts each positive example but no negative ones. The background knowledge is a set of clauses that is not changed during the learning process. All the examples are ground atoms given in the following form:

```
positive cardiac_cycles([p_peak,n_peak,line,p_peak,par,...],[]).
...
negative cardiac_cycles([p_peak,p_peak,n_peak,line,p_peak,...],[]).
```

where the following notations are used:

- *n_peak* denotes a peak with negative peak extremum,
- *p_peak* denotes a peak with positive peak extremum,
- *line* denotes a straight line primitive segment,
- *par* denotes the parabola.

The whole input for the system can be seen in Appendix 6.1.

With the help of this system not only the place of the error can be located in the program but in many cases starting from an incorrect classifier program we can infer a *correct* program using unfolding transformations and the IDT debugger. The structure of the system can be seen in Figure 7.

4 The learning process

In the following we explain how the integrated system refines an initial DCG representation of a waveform. As an example a cardiac cycle description is selected from the whole ECG waveform description. Every cardiac complex contains one or more cardiac waves, but the most important phase in the heart action is the QRS complex. This part was chosen to demonstrate the learning process. In the

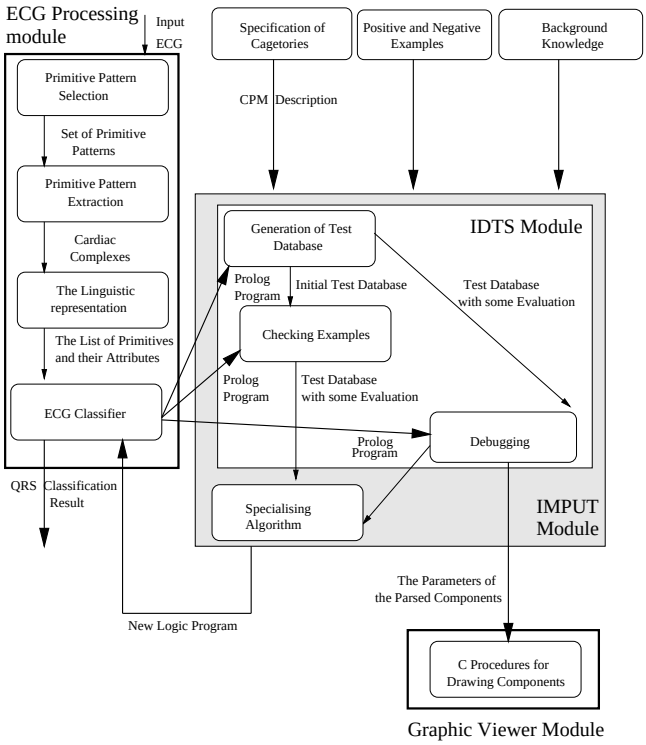

Fig. 7. The structure of the extended PECG system

original description a QRS complex can be built up from one to seven peaks and these clauses match the QRS pattern. We tried to refine this initial DCG program to recognize the correct series of peaks in the QRS complex.

The input description contains the program for classifying cardiac cycles, the background knowledge, the positive and negative examples (see Appendix 6.2). The task of our integrated system was to find the correct program which can recognize only the positive examples.

The main issues of the session are the following. The user starts the program by entering `start` then enters the name of the file to be processed. All clauses read are numbered. The clauses obtained by unfolding are numbered by pairs of numbers `i-j` where the first number denotes the parent, the second denotes the substituted clause.

```
| ?- start.
Welcome to IMPUT learning system.
Please enter the filename to be processed: cardiac_cycle_description.

 The background knowledge is:
17: t_or_p(A,B):-C(A,p_peak,C),C(C,n_peak,B)
```

```
...
24: segment(A,B):-C(A,par,B)

 The theory needs to be specialized is:
0: cardiac_cycles-->cardiac_cycle(A,C),cardiac_cycles(C,B)
1: cardiac_cycles-->B=A
2: cardiac_cycle-->qrs(A,C),non_qrs(C,B)
3: qrs-->peak(A,C),peaks(C,B)
...
16: p(A,B):-t_or_p(A,B)

 The positive examples are:
25: cardiac_cycles([p_peak,n_peak,line,...])
...
36: cardiac_cycles([n_peak,p_peak,n_peak,...])

 The negative examples are:
37: cardiac_cycles([p_peak,p_peak,line,...])
...
76: cardiac_cycles([n_peak,n_peak,p_peak,...])
```

The positive and negative examples are checked. At the beginning, both the positive and negative examples have to be covered by the initial description. Firstly, the system checks if the sets of positive and negative examples are distinct, secondly if there exist clauses which are not necessary to deduce the positive examples. If such clauses are exist then those will be removed from the program and the remaining clauses are listed. In this example clauses $7, 8, 9, 11, 12, 14, 15, 16$ are removed.

After that PECG checks for the negative examples. If the system finds a negative example covered by the initial program then the learning phase is invoked, otherwise the system has obtained the correct program. Only the covered negative examples are displayed.

The first step of the learning is to find the clause to be unfolded by means of the debugger. The questions asked by the system can be answered with *yes* or *no*. The answer *no* means that the debugger has found the false clause. In this case the debugging process is finished and the unfolding process can be started. In the opposite case the debugging goes on to the next predicate. In this example the last question is

```
 Is it ok [qrs([p_peak,p_peak])] (y/n) n
```

The answer is *no* because in the correct classification a positive peak can be followed only by negative peak.

```
37: cardiac_cycles([p_peak,p_peak,line,...[]) covered.
...
76: cardiac_cycles([n_peak,n_peak,p_peak,],[]) covered.

The fact cardiac_cycles([p_peak,p_peak,line,...],[]) is covered by the theory.
Starting the false proc. algorithm to determine the basis of the unfolding.
```

```
Is it ok [peaks([])] (y/n) y
Is it ok [peaks([p_peak])] (y/n) y
Is it ok [qrs([p_peak,p_peak])] (y/n) n],
```

The most complex part of the system is the unfolding part. As mentioned earlier, it is supposed that if a negative example is covered by the current version of the initial program, then there is at least one clause which is responsible for this incorrect covering. We have found that the clause $\texttt{peaks} --> \texttt{peak}(\texttt{A},\texttt{C}),\texttt{peaks}(\texttt{C},\texttt{B})$ is incorrect. This clause cannot be removed from the initial program because the derivation of a positive example contains this clause.

```
Unfolding at the clause instance:
   3: qrs-->peak([p_peak,p_peak,line,...],[p_peak,line,p_peak,...]),
            peaks([p_peak,line,p_peak,...],[line,p_peak,par,...])

 - trying resolvent(s): [3-1] actual minimum is: 12.5101.
 - trying resolvent(s): [3-2] actual minimum is: 0.

The result of the unfolding is:

0: cardiac_cycles-->cardiac_cycle(A,C),cardiac_cycles(C,B)
1: cardiac_cycles-->B=A
2: cardiac_cycle-->qrs(A,C),non_qrs(C,B)
3-4: qrs(A,B):-peak(A,C),peak(C,D),peaks(D,B)
3-5: qrs(A,B):-peak(A,C),B=C
...
```

The new program will not cover the first negative examples, but covers the other ones. By repeatedly using the debugger we can find an other erroneous clauses. The last step of the learning process is the following:

```
Checking positive examples:

Checking negative examples:

66: cardiac_cycles([n_peak,p_peak,n_peak,...],[]) covered.

 The above theory:
    covers   12 positive samples from 12 (100.00%) and
    fails on 39 negative samples from 40 (97.50%).

 The fact cardiac_cycles([n_peak,p_peak,n_peak,...],[]) is covered by the theory.
 Starting the false proc. algorithm to determine the basis of the unfolding.

 Unfolding at the clause instance:
 3-4-4-22-22-4-21-4-21-4-22-21-4:
 qrs([n_peak,p_peak,n_peak,...],[line,p_peak,par,...]):-
   C([n_peak,p_peak,n_peak,...],n_peak,[p_peak,n_peak,p_peak,...]),
   C([p_peak,n_peak,p_peak,...],p_peak,[n_peak,p_peak,n_peak,...]),
   C([n_peak,p_peak,n_peak,...],n_peak,[p_peak,n_peak,p_peak,...]),
   C([p_peak,n_peak,p_peak,...],p_peak,[n_peak,p_peak,p_peak,...]),
```

```
   C([n_peak,p_peak,p_peak,...],n_peak,[p_peak,p_peak,line,]),
   C([p_peak,p_peak,line,...],p_peak,[p_peak,line,p_peak,...]),
   peak([p_peak,line,p_peak,...],[line,p_peak,par,...]),
   peaks([line,p_peak,par,...][line,p_peak,par,...])

 - trying resolvent(s): [3-4-4-22-22-4-21-4-21-4-22-21-4-7] actual minimum is: 3.44203.
 - trying resolvent(s): [3-4-4-22-22-4-21-4-21-4-22-21-4-8] actual minimum is: 3.52546.

 The result of the unfolding is:

0: cardiac_cycles-->cardiac_cycle(A,C),cardiac_cycles(C,B)
1: cardiac_cycles-->B=A
2: cardiac_cycle-->qrs(A,C),non_qrs(C,B)
3-4-4-21-21-4-4-22-4-22-4-21: qrs(A,B):-
  C(A,p_peak,C),C(C,n_peak,D),C(D,p_peak,E),C(E,n_peak,F),
  C(F,p_peak,G),C(G,n_peak,H),C(H,p_peak,I),peaks(I,B)
...

 Checking positive examples:

 Checking negative examples:

 The above theory:
    covers   12 positive samples from 12 (100.00%) and
    fails on 40 negative samples from 40 (100.00%).
```

The learning process is finished because the program above has covered all positive examples and failed on all negative ones. The complete result can be found in Appendix 6.2.

5 Conclusion, Further Work

In this paper an integrated system has been presented which can classify and learn ECG waveforms. Combining the IMPUT method with the PECG system the integrated system was able to suggest correct clauses to replace the *buggy* clauses recognized during the debugging process.

In general the graphic viewer helps to find bugs, and in our case shows the ECG subpattern is currently being analyzed. This module was implemented in C, the C routines are UNIX portable and work together with the Prolog interpreter.

The main advantage of our system comparing with Skordalakis' approach is that our integrated tool can positively assists the developer (user) working with the ECG classifier program because the program itself is "self-correcting".

Applying the IMPUT learning method to the ECG domain we found that correct DCG programs can be inferred for the basic structures of complex ECG waveforms. In these approaches the user only needs to prepare very general initial DCG programs. The extended PECG system is able to specialise these programs into the correct ones in a very effective way. However, our general experiences has

shown for us that this method is not really suitable for the learning rules which describe the computation of the values of the arguments of these structures. For this task we are going to try to apply other ILP techniques (Progol [11], LINUS [5], CLAUDINE [4]). In general it is concluded that the effective employment of the ILP method for real like applications may require the combination of different ILP techinques.

The current version of the PECG system is about 3500 lines in SICStus Prolog, and 500 lines written in C for the implementation of the graphic viewer. The IDT module is about 1400 Prolog lines, containing 150 predicates while the classifier program is more than 2000 lines in Prolog. The IMPUT system itself contains more than 800 lines without the IDT module. [5]

We are going to extend our system in the following way:

At present the PECG system does not contain any preprocessing module for the linguistic representation of ECG waveforms. Therefore we could test the performance of our system with only preprocessed data presented in the literature [20]. To make our system a complete ECG recognizer we plan to implement these preprocessing modules. We also plan to ensure an interactive connection between the graphic viewer and the user. It means that the user may control the analyzing process from the graphic window by marking the subparts of the waveforms to be analysed.

At present, the PECG system is able to analyze a large set of ECG waveforms. We plan to apply ILP methods to transformations of the classifier program, such that the transformed programs can recognize further ECG waveforms appearing in actual clinical situations. We are working on this task in conjunction with ECG medical experts from the Medical University in Szeged.

References

1. Alexin, Z., Gyimóthy, T., Boström, H.: Integrating Algorithmic Debugging and Unfolding Transformation in an Interactive Learner in Proceedings of the 12th European Conference on Artificial Intelligence ECAI-96 ed. Wolfgang Wahlster, Budapest, Hungary (1996) 403–407 John Wiley & Son's Ltd. 1996.
2. van Bemmel J. H., Willems, J. L: Trends in Computer Processed Electrocardiograms North-Holland, Amsterdam (1976)
3. Boström, H., Idestam-Almquist, P.: Specialization of Logic Programs by Pruning SLD-trees., Proc. of the Fourth International Workshop on Inductive Logic Programming (ILP-94) Bad Honnef/Bonn Germany September 12-14. (1994) 31–47
4. De Raedt, L., Bruynooghe, M.: A theory of clausal discovery Proc. of the 13th International Joint Conference on Artificial Intelligence Morgan Kaufmann (1993)
5. Džeroski, S., Lavrač, N.: Inductive Learning in Deductive Databases, IEEE Transactions on Knowledge and Data Engineering 5 (6): (1994) 939–949
6. Fritzson, P., Gyimóthy, T., Kamkar, M., Shahmeri, N.: Generalized Algorithmic Debugging and Testing in Proceedings of ACM SIGPLAN '91 Conference on Pro-

[5] The implementation language currently is SICStus Prolog Version 3.0 #3 running on a SUN SPARCStation.

gramming Language Design and Implementation, Toronto, Ontario 1991. ACM SIGPLAN Notices 26, 6, (1991) 317–326
7. Kókai, G., Alexin, Z., Kocsis, F.: The IDT System and its Application for Learning Prolog Programs. Proc. of the Sixth International Conference on Artificial Intelligence and Information Control Systems of Robots (AIICSR-94) Smolenice Castle Slovakia September 12-16. (1994) 315–320
8. Kókai, G., Alexin, Z., Gyimóthy, T.: Classifying ECG Waveforms in Prolog Proc. of the Fourth International Conference on The Practical Application of PROLOG (PAP96) London, United Kingdom April 23-25, (1996) 193–221
9. Antti Koski: A semantic-syntactic Recognition System Based on Attribute Automata Proceedings of the Third Symposium on Programming Languages and Software Tools Kääriku, Estonia August 23-24 1993 (83-94)
10. Lavrač, N., Džeroski, S.: Inductive Logic Programming: Techniques and Applications Ellis Horwood, (1994)
11. Muggleton, S.: Inverse entailment and Progol, New Generation Computing Vol 13. (special issue on Inductive Logic Programming) Ohmsha, (1995) 245–286
12. Muggleton, S., De Raedt, L.: Inductive Logic Programming: Theory and methods, Journal of Logic Programming 19 (20) (1994) 629–679
13. Ostrand, T. J., Balker, M. J.: The Category-Partition Method for Specifying and Generating Functional Tests CACM 31:6 June (1988) 676–686
14. Paakki, J., Gyimóthy, T., Horváth T.,: Effective Algorithmic Debugging for Inductive Logic Programming. Proc. of the Fourth International Workshop on Inductive Logic Programming (ILP-94) Bad Honnef/Bonn Germany September 12-14. (1994) 175–194
15. Shapiro, E. Y.: Algorithmic Program Debugging MIT Press (1983)
16. Skordalakis, E.: ECG Analysis in Syntactic and Structural Pattern Recognition Theory and Applications ed. Bunke, H. and Sanfeliu, A. World Scientific (1990) 499–533
17. Skordalakis, E., Papakonstantinou G.: Towards an Attribute Grammar for the Description of ECG Waveforms 7th International Conference on Pattern Recognition (1984).
18. Tamaki, H., Sato, T.: Unfold/Fold Transformations of Logic Programs, Proceedings of the Second International Logic Programming Conference, Uppsala University, Uppsala, Sweden (1984) 127–138
19. Willems, J. L., van Bemmel, J. H., Zywietz, C.: Computer ECG Analysis: Towards Standardization North-Holland, Amsterdam (1986)
20. Willems, J. L., et al.: Establishment of reference library for evaluating computer ECG measurements programs Computers Biomedical Resources 18 (1985) 439-457

6 Appendix

6.1 The input for the extended PECG

```
0: cardiac_cycles --> cardiac_cycle, cardiac_cycles.
1: cardiac_cycles --> {true}.

2: cardiac_cycle  --> qrs, non_qrs.

3: qrs     --> peak,peaks.
```

```
4: peaks   --> peak,peaks.
5: peaks   --> {true}.

6: non_qrs --> sr.
7: non_qrs --> interwave_segment, t, interwave_segment .
8: non_qrs --> interwave_segment, p, interwave_segment .
9: non_qrs --> interwave_segment, t, interwave_segment, p,
               interwave_segment.

10: sr  --> segment, interwave_segment.
11: sr  --> peak, interwave_segment.

12: interwave_segment --> segment, interwave_segment.
13: interwave_segment --> {true}.
14: interwave_segment --> peak, interwave_segment .

15: t  --> t_or_p.
16: p  --> t_or_p.

17: background (t_or_p  --> [p_peak], [n_peak]).
18: background (t_or_p  --> [n_peak], [p_peak]).
19: background (t_or_p  --> [p_peak]).
20: background (t_or_p  --> [n_peak]).
21: background (peak    --> [p_peak]).
22: background (peak    --> [n_peak]).
23: background (segment --> [line]).
24: background (segment --> [par]).

25: positive cardiac_cycles([p_peak,n_peak,line,...],[]).
26: positive cardiac_cycles([n_peak,p_peak,line,...],[]).
27: positive cardiac_cycles([n_peak,p_peak,n_peak,...],[]).
28: positive cardiac_cycles([p_peak,n_peak,p_peak,...],[]).
...
37: negative cardiac_cycles([p_peak,p_peak,line,...],[]).
38: negative cardiac_cycles([n_peak,n_peak,line,p_peak,...],[]).
39: negative cardiac_cycles([p_peak,p_peak,n_peak,...],[]).
40: negative cardiac_cycles([n_peak,n_peak,p_peak,...],[]).
...
```

6.2 The output of the extended PECG

```
0: cardiac_cycles --> cardiac_cycle,cardiac_cycles
1: cardiac_cycles --> {true}
2: cardiac_cycle  --> qrs,non_qrs
3-4-4-21-21-4-4-22-4-22-4-21: qrs -->
  [p_peak],[n_peak],[p_peak],[n_peak],[p_peak],[n_peak],[p_peak],peaks

3-4-4-21-21-4-4-22-4-22-5: qrs -->
  [p_peak],[n_peak],[p_peak],[n_peak],[p_peak],[n_peak]
```

```
3-4-4-21-21-4-4-22-5-21: qrs -->
  [p_peak],[n_peak],[p_peak],[n_peak],[p_peak]

3-4-4-21-21-4-5-22: qrs -->
  [p_peak],[n_peak],[p_peak],[n_peak]

3-4-4-21-21-5: qrs --> [p_peak],[n_peak],[p_peak]

3-4-4-22-22-4-21-4-21-4-22-21-4-22: qrs -->
  [n_peak],[p_peak],[n_peak],[p_peak],[n_peak],[p_peak],[n_peak],peaks

3-4-4-22-22-4-21-4-21-4-22-21-5: qrs -->
  [n_peak],[p_peak],[n_peak],[p_peak],[n_peak],[p_peak]

3-4-4-22-22-4-21-4-21-5-22: qrs -->
  [n_peak],[p_peak],[n_peak],[p_peak],[n_peak]

3-4-4-22-22-4-21-5: qrs -->
  [n_peak],[p_peak],[n_peak],[p_peak]

3-4-4-22-22-5: qrs --> [n_peak],peak,[n_peak]
3-4-5-21-22: qrs --> [n_peak],[p_peak]
3-4-5-22-21: qrs --> [p_peak],[n_peak]
3-5: qrs --> peak
5: peaks --> {true}
6: non_qrs --> sr
10: sr --> segment,interwave_segment
13: interwave_segment --> {true}
```

This article was processed using the LaTeX macro package with LLNCS style