Learning Biomedical Patterns

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

[1] Institute of Informatics, József Attila University
Árpád tér 2, H-6720 Szeged, Hungary
Phone: (36) +(62) 311184, Fax: (36) +(62) 312292
e-mail: kokai@inf.u-szeged.hu

now visiting the Chair of Programming Languages,
Department of Computer Science,
Friedrich-Alexander University of Erlangen-Nürnberg
Martenstr. 3. D-91058 Erlangen, Germany
e-mail: kokai@informatik.uni-erlangen.de

[2] Department of Applied Informatics, József Attila University
Árpád tér 2, H-6720 Szeged, Hungary
Phone: (36) +(62) 311184, Fax: (36) +(62) 312292
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
Phone: (36) +(62) 454139, Fax: (36) +(62) 312508
e-mail: gyimi@inf.u-szeged.hu.

Abstract. In this paper a learning system is presented that is able to classify biomedical patterns grammatically described as a combination of primitives. The system helps the user to correct the classifier program if it cannot recognize the given input waveforms. The application of the system for learning biomedical waveforms is demonstrated by using the carotid pulse waveform as an example. [4]

1 Introduction

In this paper a learning system is presented that is able to classify biomedical patterns grammatically described as a combination of primitives. The system helps the user to correct the classifier program if it cannot recognize the given input waveforms. The application of the system for learning biomedical waveforms is demonstrated by using the carotid pulse waveform as an example.

The system presented in [5] integrates an electrocardiogram (ECG) waveform classifier (called PECG) with an interactive learner (called IMPUT). In this paper we show that not only ECG waveforms but biomedical waveforms can also be learnt with the help of this system. We can say, if an initial pattern grammars

[4] This work was supported by the grant OTKA T14228 and the project PHARE TDQM 9305-02/1022 ("ILP2/HUN").

can be formulated for classifying these input waveforms then our system is able to assist in the improving process of these grammars.

The basic idea of PECG, introduced in [4], is to integrate the classifier program (implemented in Prolog) with the Interactive Diagnosis and Testing (IDT) module [9] and a 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 built-in IDT debugger module is used to find the false clause.

The learning part of the PECG is based on an Inductive Logic Programming (ILP) [7] method called IMPUT introduced in [1]. The IMPUT system combines the interactive debugging technique IDT with the specialization algorithm SPECTRE presented in [2].

The new method involves an integration of PECG with the IMPUT interactive learning system. 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 a *buggy* Prolog program we can infer a *correct* program using an unfolding transformation and the IDT debugger. Applying the IMPUT method, the extended PECG system is able to suggest a correct solution to the user to replace the buggy clause recognized during the debugging process.

The main result presented in this paper is that a system is developed for a real size Prolog application. The IMPUT system can effectively assist in preparing the correct description of the basic structures of biomedical waveforms more specifically ECG waveforms and carotid pulse waveforms.

Sections 2 contains, a short descriptions of the PECG, the IMPUT and the integrated systems. In Section 3 the operation of the integrated system is demonstrated with the help of a carotid pulse waveform description. Finally, in Section 4 a brief summary and comments on possible future studies are presented.

2 The architecture of the 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 [4], [1]. The remained part of this section deals with the description of the integrated system.

2.1 The PECG system

The original PECG system contains three main parts: the ECG processing, the IDT, and the graphic viewer modules. It is based on an attribute grammar approach of ECG waveform analysis published by Skordalakis [12].

The ECG processing module is built up from four additional submodules. In the *primitive pattern selection* part an ECG waveform can be considered as a composite pattern which can be decomposed as soon as possible into primitive patterns. The main task of the *primitive pattern extraction* is to select the peaks. In the *linguistic representation* part the result of the last two steps are represented as a string of symbols from an alphabet. The *ECG classifier* module recognizes the ECG waveforms from their linguistic representations. The ECG classifier part was extended such that the new system is able to handle carotid pulse waveforms introduced in [6], [13].

The IDT part uses a method based on the algorithmic program debugging technique introduced by Shapiro [11]. 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* [3], with the algorithm introduced in [11]. 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 [9] and a similar system for imperative languages was introduced in [3].

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 uses the test results to avoid asking the user with "difficult" questions.

In the PECG system the algorithmic debugger IDT has been extended by adding a graphic viewer. This graphic module shows the ECG waveform being analyzed and in this way provides effective assistance for the user to answer the questions invoked by the debugger.

2.2 The IMPUT system

The IMPUT system consists of two main parts. The specialization algorithm comes from the SPECTRE system while the 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. This modification initiated some further improvements and these are incorporated in IMPUT as well.

The algorithm SPECTRE [2] specializes logic programs with respect to positive and negative examples by applying the transformation rule *unfolding* [14] together with clause removal. The choice of which literal to unfold upon is controlled by a computation rule, which is given as input to the algorithm. A detailed description of SPECTRE can be found in [2].

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.

2.3 Integration of PECG and IMPUT system

The extended PECG system involves an integration of PECG with the IMPUT interactive learning system. 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 a *buggy* Prolog program we can infer a *correct* program using an unfolding transformation and the IDT debugger. Applying the IMPUT method, the extended PECG system is able to suggest a correct solution to the user replacing the buggy clause recognized during the debugging process with its resolvents. The whole structure of the system can be seen in Figure 1.

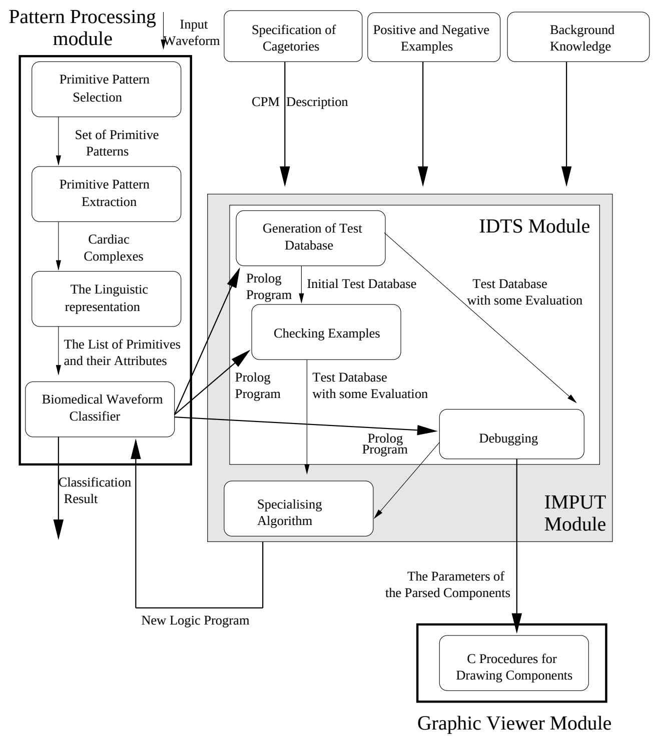

Fig. 1. The structure of the extended PECG system

3 Experimental result

In this section we shall explain how the integrated system can learn DCG rules for a biomedical waveform descriebed by a grammar. In Skordalakis's papers [6], [12] always an attribute grammar was presented for recognition of waveforms. As an example a carotid pulse waveform description is selected to demonstrate the learning process. The syntactic part of the grammar in Prolog is presented in Appendix 5.1. Skordalakis's system contains a preprocessing module which produces a linguistic representation from an ECG signal. In each case this linguistic representation is used as the input for our system.

Our approach starts from attribute grammar specifications of waveforms, however we use definite clause grammar (DCG) representations for the recognition of these waveforms. The difference between an attribute grammar representation and a Prolog program made the rewriting process non-trivial. As it was mentioned earlier, the system was extended with a graphic viewer to trace the history of the traversing of the proof tree of the analyzer program, but to keep our example as simple as possible here we do not present the results any of these displays. In the following we present how the IMPUT learning system refines an initial DCG program. To demonstrate the work of IMPUT two bugs were introduced in the grammar presented in Appendix 5.1.

- A new rule for the nonterminal `m3` was added to the description;

  ```
  m3-->pek
  ```

- The `cap` terminal was changed to `m3` nonterminal in the following DCG rules.

  ```
  pos_wave-->m3
  pos_wave-->m3,large_neg
  ```

The input description contains the initial program for recognition carotid pulse waveforms, the background knowledge, the positive and negative examples (see Appendix 5.2). This modified program accepts incorrect carotid waveforms — these are the negative examples. The task of our integrated system is to find the correct program which can recognize only the positive examples. The main steps of the learning are the following. The user starts the program by entering `start` then enters the name of the file to be processed. 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: carotid_example.

 The background knowledge is:
22: upslope-->[ups]
...
34: lshold-->[lsh]

 The theory needs to be specialized is:
```

```
0: main(A,[]):-carotid_pulse_wave(A,[])
...
21: dicrotic_wave-->pek

 The positive examples are:
35: main([ups,cap,ccup,rsh,lneg,hor,tre],[])
...
43: main([ups,cap,mneg,lneg,tre],[])

 The negative examples are:
44: main([ups,rsh,ccup,rsh,lneg,tre],[])
...
47: main([ups,rsh,lneg,mneg,lneg,tre],[])
```

The positive and negative examples are checked. At the beginning, both positive and negative examples may 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 derive the positive examples. If such clauses are exist then those will be removed from the program. In this case the program found that the following clauses

```
pos_wave --> m3, large_neg
m3       --> pek
```

are superfluous in the description. It means that the extended PECG system recognized the first error which we made. The superfluous clauses were left out and the remaining clauses were listed.

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 [pos_wave([rsh])] (y/n) n
```

The answer for the second question is *no* because in the correct classification the `pos_wave` can contain only `cap` values. The reason of this error is the modification which allows the derivation of `m3` from `pos_wave`.

```
 Checking negative examples:
44: main([ups,rsh,ccup,rsh,lneg,tre],[]) covered.
46: main([ups,rsh,mneg,lneg,pek,tre],[]) covered.

 The fact main([ups,rsh,ccup,rsh,lneg,tre],[]) is covered by the theory.
 Starting the false proc. algorithm to determine the basis of the unfolding.
```

```
Is it ok [m3([rsh])] (y/n) y.
Is it ok [pos_wave([rsh])] (y/n) n
```

The most complex module 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 pos_wave --> m3 is incorrect. This clause cannot be removed from the initial program because the derivation of a positive example contains this clause. The resolvents of this clause are: pos_wave --> cap and pos_wave --> rshold. The second clause is removed, because there is no positive example whose derivation contains it. The system displays the clause to be unfolded and the result of the unfolding.

```
 Unfolding at the clause instance:
   8: pos_wave-->m3([rsh,ccup,rsh,lneg,tre],[ccup,rsh,lneg,tre])

 - trying resolvent(s): [8-1] actual minimum is: 4.18789.

 The result of the unfolding is:
0: main(A,[]):-carotid_pulse_wave(A,[])
...
8-11: pos_wave(A,B):-cap(A,B)
...

 Checking positive examples:

 Checking negative examples:

 The above theory:
    cover   9 positive samples from 9 (100.00%) and
    fail on 4 negative samples from 4 (100.00%).
```

The learning process is finished because the program above covers positive examples and failed on all negative ones. The complete result can be found in Appendix 5.3.

4 Conclusion, Further Work

In this paper an integrated system has been presented which can classify and learn different biomedical waveforms. Combining the IMPUT method with the PECG system the integrated system is able to suggest correct clauses by replacing buggy clauses recognized during the debugging process.

The graphic viewer helps to find bugs, and in our case shows the 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 classifier program because the program itself is "self-correcting".

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 different kind of biomedical waveforms. To make our system complete we plan to implement this one. 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 analyzed.

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. 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
3. Fritzson, P., Gyimóthy, T., Kamkar, M., Shahmeri, N.: Generalized Algorithmic Debugging and Testing in Proceedings of ACM SIGPLAN '91 Conference on Programming Language Design and Implementation, Toronto, Ontario 1991. ACM SIGPLAN Notices 26, 6, (1991) 317–326
4. 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
5. Kókai, G., Alexin, Z.,Gyimóthy, T.: Analyzing and learning ECG Waveforms submitted to The Sixth International Workshop on Inductive Logic Programming (ILP'96) 28-30 August, 1996 Stockholm, Sweden
6. Moratis, C., Papakonstantinou, G., Skordalakis, E. A Syntactic Model for Pattern Generation Proveedings Int. AMSF. Conf. "Modelling & Simulation" Athens, June 27-29, 1984, Vol. 1.2, p. 1-8
7. Muggleton, S., De Raedt, L.: Inductive Logic Programming: Theory and methods, Journal of Logic Programming 19 (20) (1994) 629–679
8. Ostrand, T. J., Balker, M. J.: The Category-Partition Method for Specifying and Generating Functional Tests CACM 31:6 June (1988) 676–686
9. 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
10. Pereira, F. C. N., Warren, D. H. D.: Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks Artificial Intelligence 13: 231-278, (1980)

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

11. Shapiro, E. Y.: Algorithmic Program Debugging MIT Press (1983)
12. Skordalakis, E.: ECG Analysis in Syntactic and Structural Pattern Recognition Theory and Applications ed. Bunke, H. and Sanfeliu, A. World Scientific (1990) 499–533
13. Stockman, G., Kanal, L., Kyle, M. C.: Structural Pattern Recognition of Carotid Pulse Wave Using a General Waveform Parsing System CACM, vol. 19, No. 12, pp. 683-695, 1976
14. 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

5 Appendix

5.1 The syntactic part of the original carotid pulse waveform description

```
1: main(Inp,[])       :-   carotid_pulse_wave(Inp,[]).
2: carotid_pulse_wave -->  systoloic, diastolic.
3: systoloic          -->  upslope, maxima, large_neg.
4: maxima             -->  m1, m2, m3.
5: maxima             -->  med_pos, m3.
6: maxima             -->  m1, med_neg.
7: m1                 -->  lshold.
8: m1                 -->  pos_wave.
9: pos_wave           -->  cap.
10: pos_wave           -->  cap, large_neg.
11: m2                 -->  ccup.
12: m2                 -->  neg_wave.
13: neg_wave           -->  vcup.
14: neg_wave           -->  vcup, large_pos.
15: m3                 -->  rshold.
16: m3                 -->  cap.
17: diastolic          -->  trailing_edge.
18: diastolic          -->  dicrotic_wave, trailing_edge.
19: dicrotic_wave      -->  cap.
20: dicrotic_wave      -->  hor.
21: dicrotic_wave      -->  pek.
22: upslope            -->  [ups].
23: large_pos          -->  [lpos].
24: large_neg          -->  [lneg].
25: med_pos            -->  [mpos].
26: med_neg            -->  [mneg].
27: trailing_edge      -->  [tre].
28: hor                -->  [hor].
29: cap                -->  [cap].
30: pek                -->  [pek].
31: ccup               -->  [ccup].
32: vcup               -->  [vcup].
33: rshold             -->  [rsh].
34: lshold             -->  [lsh].
```

5.2 The input description for our system

```
background (upslope       --> [ups] ).
background (large_pos     --> [lpos]).
background (large_neg     --> [lneg]).
background (med_pos       --> [mpos]).
background (med_neg       --> [mneg]).
background (trailing_edge --> [tre] ).
background (hor           --> [hor] ).
background (cap           --> [cap] ).
background (pek           --> [pek] ).
background (ccup          --> [ccup]).
background (vcup          --> [vcup]).
background (rshold        --> [rsh] ).
background (lshold        --> [lsh] ).

positive main([ups,cap,ccup,rsh,lneg,hor,tre],[]).
positive main([ups,cap,mneg,lneg,cap,tre],[]).
positive main([ups,lsh,ccup,rsh,lneg,tre],[]).
positive main([ups,lsh,ccup,rsh,lneg,hor,tre],[]).
positive main([ups,lsh,ccup,rsh,lneg,pek,tre],[]).
positive main([ups,cap,vcup,cap,lneg,tre],[]).
positive main([ups,lsh,mneg,lneg,pek,tre],[]).

negative main([ups,rsh,ccup,rsh,lneg,tre],[]).
negative main([ups,rsh,lneg,ccup,rsh,lneg,hor,tre],[]).
negative main([ups,rsh,mneg,lneg,pek,tre],[]).
negative main([ups,rsh,lneg,mneg,lneg,tre],[]).
```

5.3 The output of the extended PECG system

```
0: main(A,[]):-carotid_pulse_wave(A,[])
1: carotid_pulse_wave-->systoloic,diastolic
2: systoloic-->upslope,maxima,large_neg
3: maxima-->m1,m2,m3
5: maxima-->m1,med_neg
6: m1-->lshold
7: m1-->pos_wave
8-11: pos_wave(A,B):-cap(A,B)
10: m3-->rshold
11: m3-->cap
13: m2-->ccup
14: m2-->neg_wave
15: neg_wave-->vcup
17: diastolic-->trailing_edge
18: diastolic-->dicrotic_wave,trailing_edge
19: dicrotic_wave-->cap
20: dicrotic_wave-->hor
21: dicrotic_wave-->pek
```