USING SEEK FOR MULTI-CHANNEL
PATTERN RECOGNITION

Kenneth P. Birman

TR 82-529
October 1982

Department of Computer Science
Cornell University
Ithaca, New York 14853
USING SEEK FOR MULTI-CHANNEL PATTERN RECOGNITION

Kenneth P. Birman

Department of Computer Science, Cornell University

Address for proofs and correspondence:

Prof. Kenneth P. Birman
Department of Computer Science
Cornell University
Ithaca, New York 14853

(607) 256-4052

Running title:

MULTI-CHANNEL PATTERN RECOGNITION
Abstract

Our work on computerized analysis of the 2-channel, 24-hr electrocardiogram has resulted in the development of multi-channel signal processing systems that learn by observation. In this paper a new tool for implementing such algorithms is described: the pattern recognition language SEEK. Programs written in SEEK build a knowledge base containing tree-like data structures, each of which stores acquired information about a particular multi-channel waveform. Input data is interpreted by performing an efficient parallel evaluation of the structures in the knowledge base. Our work is applicable to a wide variety of pattern recognition problems that arise in medical signal processing. The approach is illustrated with examples drawn from ECG analysis.
1. Introduction

During the past decade pattern recognition algorithms have been proposed for a variety of medical computing problems. Frequently, the software systems implemented to solve these problems are large and specialized, obscuring any algorithms that are of general interest. A result is that medical pattern recognition algorithms are often inaccessible to researchers studying related problems.

In our investigation of rule-based knowledge representation and its application to long-term ECG monitoring we were faced with the problem of specifying multi-channel pattern recognition algorithms in a "high-level" form [1] [2]. Our solution was to design a new programming language, SEEK, which resembles existing programming languages but has special features well suited to implementation of pattern recognition algorithms for signal processing. SEEK permits a programmer to construct parallel recognizers that simultaneously scan for instances of several different objects. SEEK has evolved from a simple experimental language into a powerful tool suitable for preparation of substantial signal processing programs.

We have achieved several objectives by using SEEK to build signal processing systems. First, SEEK provides a methodology for decomposing multi-channel pattern-recognition problems into less complex sub-problems. This facilitates design of programs that perform pattern recognition in complex signals exhibiting unpredictable artifact in one or more channels. Secondly, programs written in SEEK learn by observation. This feature proved critical during development of computer systems that avoid systematic errors during long-term signal analysis. Finally, SEEK programs are expressed in a form that is easily modified. This simplifies program development, allows ease of maintenance, and facilitates experimentation with new algorithms.

It should be stressed that we use the term learning to denote the process whereby a computer systems automatically acquires information and makes effective use of the resulting knowledge base. In particular, if a program written in SEEK interacts with a human while learning, that human is not expected to have a detailed understanding of the internal structure of the system. Instead, the human functions as an "expert" who can answer difficult questions that the
computer system is unable to resolve. This can be contrasted with work in which the human operator of a computer system must understand precisely how the system works.

In the problems we consider, knowledge acquisition is constrained by a-priori rules. In effect, software systems that learn do so within a pre-established framework. The rules that define this framework are specified by the system designer, and take the form of a program written in a specialized programming language. This strategy of combining precise control structures with knowledge acquired by observation represents one major contribution of our research. Although similar work has been done by some researchers in artificial intelligence [3] [4], our methods differ because they involve the dynamic creation of new data structures to represent knowledge. This technique has not previously been combined with a "rule-based" analysis strategy. In addition, we address specialized subproblems that arise during multi-channel signal processing, such as representing multichannel waveforms and learning to interpret signals that exhibit dropouts and noise bursts on some channels.

Our work can also be contrasted with other multi-channel signal analysis efforts. Successful systems have been developed for some specific applications, notably the analysis of 8 and 12-lead ECG, with standardized lead positions [5] [6]. Recently, the 2-channel ECG problem (more difficult because lead positions are unpredictable) has received recent attention, although contemporary systems for this application operate primarily by scanning a single channel, switching channels in the event of noise, and otherwise extracting features from the second channel in the event of uncertainty about a waveform in the first [7] [8]. In contrast, our systems scan all channels simultaneously. Other differences will be described in section 2.

We begin with a discussion of rule-based knowledge representation and pattern recognition, the approach to multi-channel learning that was developed in [1] [2]. Our work draws from other research, which we review. A detailed description of the SEEK language is then presented, using examples from 2-channel ECG research. The present implementation status of the language is also described. Finally, the characteristics of other problems to which SEEK can be applied are discussed.
2. Learning About Signals and Images

This section discusses features that are common to most learning algorithms and situations in which such algorithms are necessary. Techniques that are commonly employed by signal processing systems are then reviewed. Finally, our rule-based learning methodology is described.

2.1. Systematic error and learning algorithms

The importance of constructing software systems that learn is not immediately evident in some applications because relatively simple algorithms often suffice to solve a particular problem. In such situations, the greater complexity of learning algorithms is an impediment to their use. Nonetheless, systems that cannot learn by observation risk making errors. For example, contemporary single and dual-channel ECG analysis and monitoring systems employ algorithms that define the appearance of plausible QRS waveforms, the recorded form of a heart beat. Examination of typical clinical ECG signals reveals that simple a-priori rules for QRS detection are unlikely to be reliable. One reason is that wide variability in the shape of QRS waveforms is observed from patient to patient. Moreover, QRS waveforms may vary slowly within a single recording, exhibiting occasional notches, height variations (due to respiration or patient motion), and large P-waves (due to atrial contraction) or T-waves (due to ventricular repolarization after a contraction). In figure 1 we have indicated a QRS waveform and its T-wave.

Many patients with cardiac disease experience premature ventricular contractions (PVCs). These induce waveforms that are implausible as normal QRS complexes. A single electrocardiogram may contain instances of more than one type of PVC, each corresponding to a different waveform shape on the ECG (often, PVC shape differences are not apparent unless at least two ECG channels are examined). In a typical clinical ECG one to four PVC shapes are present in addition to the normal QRS waveforms. These PVC waveforms can occur as often as thousands of times each hour.

Existing ECG scanning systems often learn QRS and PVC waveform shapes, but generally do not learn to detect QRS and PVC waveforms. Instead, they detect each QRS and PVC waveform in the ECG using a-priori methods based on assumptions about typical QRS
complexes. This strategy poses a significant risk: a QRS detection error may propagate through the shape classifier to result in misinterpretation of some segment of ECG. In particular, it is commonly assumed that some minimum interval separates successive QRS complexes, and hence that a QRS detector can safely enter an eye-closing period after finding each QRS waveform. During this period the detector does not trigger and (in principle) avoids misinterpreting any large T-waves present in the ECG. It is also commonly assumed that every QRS contains steep slopes and exceeds some minimum amplitude (this is useful if a threshold trigger is employed in the QRS detector). These assumptions are probably all reasonable if only normal ECG signals are analyzed, but are sometimes violated by precisely the abnormal cardiac rhythms that are of maximum clinical interest. For example, PVC pairs or ECG segments containing frequent PVCs may exhibit such closely spaced complexes that an eye-closing QRS detector would miss every second PVC if the usual eye-closing interval is employed for T-wave rejection. Similarly, large, late T-waves may be detected and misclassified as PVCs. A threshold detector will tend to miss small flat PVCs waveforms or, worse, to miss such waveforms but detect and misinterpret their T-waves as noise-distorted normal complexes. Cardiac disease is characterized by repetitious phenomena, hence a signal that violates a program's assumptions once will probably do so repeatedly. A result is that automatic ECG analysis systems are prone to systematic errors: ECG misclassifications that recur each time some cardiac event is encountered, censoring clinically significant data and biasing or invalidating the ECG interpretation as a whole.

Since a wide variety of systematic ECG interpretation errors can be traced to non-adaptive algorithms that rely in some fundamental way on \textit{a-priori} reasoning, we explored the development of systems that learn to analyze a particular signal by \textit{observation}. Analogous issues arise in many other medical applications. Early work [9] on the carotid pulse pressure wave involved a syntactic technique that adapts to variations in waveform shape. Rhythmic signals are well known in studies of the cardiac nervous system. Investigators of the response of nerve cells to drugs place electrode within nerves and study the neural discharge waveform. Once the electrode is placed, the waveform remains stable. Thus, a computer system that learns to interpret a single waveform can often analyze subsequent instances in a very similar manner.
In a cardiac surface mapping system, a great many ECG channels are analyzed simultaneously, each collected from a slightly different site on the cardiac surface. In addition to beat to beat similarities observed in any one channel, adjacent sites often give rise to very similar signals. A system that learns to interpret a waveform in a single channel potentially could use that knowledge to interpret similar waveforms in the immediately adjacent channels [10]. In digital angiography and echocardiography systems, a sequence of cardiac images is made during contraction of the heart. To interpret such images, a computer system initially learns the shape of the chamber by combining knowledge of chamber physiology with observation of the first image. Wall position and motion estimates help predict chamber boundaries in successive frames [11].

2.2. Signal processing methodologies

The forgoing discussion suggests that many signal and image processing systems can be viewed in terms of a single paradigm. These systems initially learn instances of objects that vary from some "expected" shape in a manner constrained by physiology and the data collection apparatus. Subsequently, data is interpreted by comparing it with the acquired knowledge base, using comparison rules that are indigenous to the application. After data is interpreted, the knowledge base is updated to improve subsequent performance and to maintain statistics.

Two approaches to signal processing have been developed and used. One employs primarily syntactic methods, while the other places greater emphasis on signal structure and uses classical time-series analysis methods to interpret waveforms. Using a rule-based methodology we have been able to integrate the two approaches and incorporate a learning mechanism.

2.2.1. Syntactic signal processing

Syntactic signal analysis encompasses a variety of algorithms that operate by encoding the signal using an alphabet of representative symbols. We can draw some examples of single-channel encodings from the ECG application. (Multi-channel encoding is accomplished by computing several single-channel encodings in parallel). One ECG encoding method begins by mapping the signal into a sequence of straight lines and slopes using a piecewise-linear interpolation algorithm.
This representation can be processed directly [12] [13] or encoded further, into a stream of symbols. For example, the alphabet \{ F, BU, SU, SD, BD \} might represent lines that are flat or sloped up or down, and with a big or small slope [14]. A waveform resembling a triangle could then be encoded as: "... F BU BD F ..." Other encodings include symbolic representation of the power of the signal in the vicinity of peaks [15] and symbolic representation of triangles extracted from the signal [16]. It is sometimes useful to consider uncertain encodings, wherein a probability or confidence estimate is associated with each symbol [17].

After encoding, waveforms will be represented by strings of symbols. If the encoding is successful, the class of strings corresponding to the waveform of interest can be recognized by searching for particular patterns. Algorithms that perform this operation are referred to as syntactic pattern recognizers. Similarly, one can define a probabilistic syntactic recognizer that searches for the pattern match having the highest probability in an encoding that associates probabilities with symbols. The theoretic aspects of these problems are well understood [13] [17] [18].

2.2.2. Structural analysis methodology

Structural signal analysis encompasses a variety of more classical signal processing techniques. Examples include matched filters or threshold triggers for waveform recognition, template correlation after alignment using centroid computations, and frequency domain analysis [19]. These methods generally assume knowledge about the content of the signal segment under analysis. Moreover, the theories underlying their use often require that the signal satisfy certain constraints. For example, stationarity is generally assumed in a frequency domain computation.

The structural approach has often been employed in ECG analysis. Systems of this sort tend to use a fixed analysis strategy based on classical signal processing techniques. For example, a threshold trigger is commonly used for QRS detection and correlation with an acquired library of waveform templates for QRS classification. Here, the term template refers to a vector of information that characterizes some waveform shape. Theoretically, template comparison should be done using a costly cross correlation, but it is common to use a simplified method to reduce cost. Specifically, the QRS waveforms are centered during template creation so that a fixed-lag
correlation can be performed. Waveforms cannot be centered using the QRS detection time because some conditions provoke erratic QRS detector behavior (small variations in the position at which the detector triggers). Consequently, most systems align waveforms around a point called the centroid. The centroid is essentially a "local" center of gravity, and is physiologically meaningless. Although centroids are more stable than QRS detectors, some signals provoke erratic centroid behavior, and in these cases fixed-lag template correlations will fail. A system that observes the performance of its centroid computations might detect this problem and resort to other, more stable, alignment methods. However, no contemporary system incorporates such adaptive behavior. Instead, frequency domain computations are sometimes used to avoid the need for precise waveform alignment [20]. Here, however, unsuccessful centering may result in clipping of the waveform near the edges of a template window or inclusion of fragments of other waveforms that should have remained outside the window. Either situation leads to contamination of the frequency spectrum.

The preceding example motivates us to conclude that systems that undertake to solve complex signal processing problems may have to combine a variety of techniques, perhaps adapting to the signal in the process of scanning it. Our rule-based signal processing methods provide a structure within which this behavior can be achieved.

2.3. Rule-based learning

Rule-based signal processing systems integrate syntactic and structural analysis methods with a learning mechanism. It should be observed that rule-based research in other areas has generally omitted learning mechanisms. Moreover, our provisions for organizing analysis methods and monitoring their performance result in greater flexibility during the system design. Finally, a rule-based scanner can entertain several possible interpretations of a signal simultaneously. A decision to reject some interpretation is generally made by a rule that has learned (acquired information) to make that decision. "Lower level" analysis operations can develop sets of hypotheses instead of reducing such sets to a single interpretation on the basis of relatively ad-hoc criteria. In effect, it is possible to postpone the resolution of conflicts until information is available with
which this can be done "intelligently."

For example, structural ECG analysis systems rely on a QRS detector that identifies each waveform using pre-established threshold and timing criteria. We have already observed that this strategy can result in systematic error. In contrast, a rule-based QRS detector can formulate a list of possible waveform locations. Although the system should subsequently reject any invalid interpretations, the rejection criteria can be based on prior observation of the signal being analyzed and not on speculation about typical signals. Our experience is that this approach is more resistant to systematic error.

Before undertaking a detailed discussion of rule-base knowledge representation, it will be helpful to consider a simple example. Figure 1 illustrates the recognition of a 2-channel QRS waveform by a rule-based analysis program that previously learned some waveform with a similar shape. The shape is represented hierarchically, using a rule-activation tree whose internal nodes are rule-activations. Rule-activations store information about various aspects of the waveform (our reasons for choosing these names will shortly be clarified). Specifically, one activation is used to represent each of the two single-channel waveforms and a third stores information that relates the two single channel waveforms. The rule-activations at the lowest level of the tree also hold a symbolic pattern that matches the single-channel waveforms to which they correspond. The symbols are drawn from the alphabet of an encoder that preprocesses the signal. Thus, as illustrated in the figure, the tree can be recognized syntactically from the bottom-up. First, syntactic matches for the two single-channel rule-activation are found, then the dual-channel rule-activation is matched.

Unlike most syntactic analysis systems, rule-based systems treat the signal encoding as a hint about content, not as a representation on which all subsequent computation must be performed. In fact, the raw (unencoded) signal can be used during the analysis in arbitrary ways. Thus, our use of a syntactic encoding does not imply a permanent loss of information. This should be contrasted with the purely syntactic approach to signal processing, in which information that is not preserved by the encoding is not available during subsequent analysis.
One can view QRS detectors as signal encoders that transform a signal into an encoded format that retains exactly one symbol for each waveform. A rule-based strategy permits the design of systems based on much richer symbolic encodings. In particular, we have found that a triangle encoding of the ECG retains enough information to reveal any waveforms that are present while simplifying the signal enough for a high-speed analysis, as illustrated in figure 1. We have also experimented with encodings based on piecewise-linear approximation, but found these encodings more difficult to work with [2].

The stages of QRS recognition are labeled in figure 1. (Note that the algorithm this figure illustrates is hypothetical and quite simple. For example, the algorithm will only recognize waveforms if both channels are noise-free.) The analysis stages are as follows:

1. The ECG signal is encoded using a triangle representation. Each symbol in the encoding has multiple attributes: a symbolic name, a start and end time, and any other information.

Figure 1: Recognition of a 2-channel QRS shape
that may be useful in the application (for example, triangle height and width).

(2) The encoded signal is scanned by a parser (pattern matcher) that searches for syntactic matches between the patterns associated with activations of the \textit{chan1} and \textit{chan2} rules and the symbolic names in the encoding. Here, \textit{chan1} matches an UP-DOWN triangle sequence and \textit{chan2} matches a single down triangle. Because the two matches are separated by a short time interval (a small \textit{skew}), the corresponding \textit{qrs} rule activation will also be matched. Now the entire tree has matched, hence detailed interpretation of the signal by the rule-activation tree can be undertaken. Note that only syntactic matching occurs during the parsing phase and that more expensive computations are postponed until strong syntactic evidence for a match has been accumulated.

(3) The \textit{chan1} and \textit{chan2} rules are invoked to perform a more detailed tests on each channel. It is determined that \textit{chan1} and \textit{chan2} matched with respective probabilities of .83 and .91. Typically, these probabilities would be determined by comparison of the attributes of the triangles detected in the encoding with information stored as part of the private knowledge base of the \textit{chan1} and \textit{chan2} rules.

(4) The pattern associated with the \textit{qrs} rule is now invoked for a detailed two-channel interpretation. A probability of .85 is obtained. At this stage, it would be normal to use more precise (and costly) comparison methods. For example, the \textit{qrs} rule might perform a frequency domain template correlation in each channel. Recall, however, that each rule-activation stores information about a single shape. Thus, any single evaluation of a 2-channel \textit{qrs} rule would do at most two template correlations.

(5) The overall probability that the rule-activation matched is now computed as the minimum of the three probabilities computed by the rule-activations: .83. This is indicative of a high degree of confidence that the shape "known" by the rule-activation matched the shape present in the ECG.

(6) Assuming that no conflicting interpretation is found, the QRS is considered to have been correctly analyzed and the knowledge associated with each rule-activation is updated. The
output expert is then invoked to take any appropriate actions based on the interpretation. We discuss the handling of conflicts shortly.

A rule-based system explores multiple interpretations of the signal in parallel. That is, if several waveform shapes are known then it is possible that matches for more than one will be found in the encoding of a single segment of the signal. User-specified timing criteria are used to make precise the notion of "single segment." These situations are called conflicts and are resolved in the following manner. When two or more rule-activation trees match a segment of the signal, the order of evaluation of the trees is determined by a conflict resolution expert (specified by the SEEK programmer). As evaluation proceeds, the conflict resolution expert acquires information about the signal, and uses this to determine that certain interpretations should be rejected. This approach permits the programmer to design a resolution strategy that learns about conflicts that occur frequently and avoids unnecessarily evaluating an interpretation that is certain to be rejected subsequently.

When a rule-based system encounters a waveform with a completely new shape, that waveform must be interpreted without using previously acquired knowledge. In the applications of interest to us this is a comparatively infrequent occurrence: artifact is a more common source of difficult signal interpretations. Therefore, special algorithms are used to identify noise and interpret new waveforms. The software that performs these functions includes much of the a-priori signal knowledge used in a rule-based analysis system. If necessary, it may maintain a private knowledge base or interact with a human supervisor. A typical interpretation expert might store private copies of templates representing each of the shapes known to the system. Relatively costly forms of template comparison could be used to explore the hypothesis that a distorted instance of some known shape has been detected.

In a rule-based signal processing system acquired information is highly integrated with signal interpretation rules. This contrasts with the strategy used in many contemporary systems, whereby the lowest level analysis makes critical decisions without using acquired information at all. The lowest level of a rule-based system computes an encoding that can retain considerable
complexity. Subsequent analysis combines general interpretation rules with specific knowledge acquired by observation. This method significantly reduces the likelihood of systematic error.

2.4. Software components of a rule-based signal processing system

We now summarize the components of a rule-based signal processing system. These systems decompose into a collection of expert subsystems. Each expert specializes in a particular aspect of the analysis problem. The notion that complex computer systems can be built from expert subsystems is a popular one in contemporary artificial intelligence research [3] [4]. Our model imposes no limitations on the design of the experts; the concept is introduced primarily because it is a particularly intuitive way to visualize large software systems.

A. Interpretation expert

This expert examines segments of the signal that cannot be processed using previously acquired information and determines a reasonable interpretation for each, maintaining a private knowledge base and possibly interacting with a human observer in difficult cases. Most such segments consist of artifact and can be rejected as such. Otherwise, if a new waveform shape has been discovered, the interpretation expert "teaches" it to the remainder of the system by requesting that a new rule-activation tree be added to the knowledge base.

B. Knowledge base:

The knowledge base is an expert that specializes in representing the past. In our systems, it consists of a collection of rule-activation trees that store knowledge about different objects. Rule activation nodes in these trees are software subroutines with private memory about particular objects, and can be evaluated to obtain a detailed interpretation of a segment of the signal. Each tree can be viewed as an expert specializing in the recognition of a particular waveform shape.

C. Rule activation:

A rule is an expert procedure for recognizing a component of a waveform. Associated with each rule is a (syntactic) pattern, and a rule is only applied to parts of the signal that match the pattern (see (D), below). Rules have symbolic names, which can appear in the patterns used by other rules. Patterns can also reference symbols used in the encoding of
any single channel. Presently, only hierachical (tree) structures of rules are supported.

Each *activated* rule manages a set of private data that characterizes the particular waveform shape that it recognizes. When a new object is learned, a corresponding rule activation is formed, binding of new instances of the private variables bound to values that characterize the object. For example, a rule to recognize triangles might define a variable to record the average height of triangles that it matches and a second variable to record their width. To store knowledge about three different triangle shapes, three activations of the triangle rule would be used, each with private copies of the variables.

Rules have four major functions. Initially, they learn new instances of an object by assigning values to the private variables in a new rule activation. During scanning, they interpret segments of the input signal to determine the probability that the segment matches the object represented by a particular rule activation. After conflict resolution they update the local variables to reflect information gained from the new instance of the object (for example, some QRS shapes vary over time). Finally, rules undertake detailed measurements of the signal for output by the analysis system.

D. Parser:

The *parser* is used to locate segments of the signal that (potentially) match existing rule-activation trees and to initiate the evaluation of those trees if a candidate match has been located. If a sufficiently long segment of the signal is scanned without finding any match, the signal interpretation expert is invoked to examine the segment and classify it as noise or a new waveform shape. To detect a match, the parser examines a symbolic encoding of the signal. The parser explores multiple signal interpretations simultaneously because it finds every match of the trees in the knowledge base with the encoded signal, even if modules overlap (conflict).

E. Encoder:

The encoding expert scans the signal and extracts a list of symbols for use by the parser. Symbols are generally primitive shapes that accurately characterize the signal, are
reasonably immune to low levels of noise, and can be computed inexpensively. The signal encoding expert operates independently from the remainder of the system and in some applications can be moved into specialized hardware devices.

F. Conflict resolver:

Conflicts are detected by the parser and are subsequently resolved under control of the conflict resolution expert (section 2.3). Resolution algorithms are application-dependent, and must be specified by the SEEK programmer. The conflict resolution expert can prevent the evaluation of any of the matching trees. It can also allow evaluation but subsequently reject one or more of the trees using probabilities estimates made by the trees in combination with other interpretation criteria known to the expert. In many applications, this expert must learn to resolve conflicts, perhaps by interacting with a human observer. We discuss this learning problem in [2]. After conflict resolution, the information managed by the trees that matched and were not rejected is updated.

G. Output:

The output expert is invoked after a segment of signal has been interpreted and all conflicts have been resolved. It obtained from application-specific stream of output data using information extracted by the rule-activation tree that matched each segment of the signal. Output information might consist of a list of times when waveforms were detected, detailed measurements from each waveform, etc. For example, an ECG analysis system normally undertakes rhythm interpretation after QRS detection and shape classification. This operation would be performed on information generated by the output expert.

2.5. Developing rule-based software systems

It would be impractical to design rule-based software without the support of specialized tools. By using SEEK, the system designer is able to specify rule-based algorithms in a simple, high level form. "Libraries" of encoding and conflict resolution expert more provide the designer with useful starting points from which specialized expert can be developed for specific applications. A complete SEEK program is constructed by building a number of independent fragments
and then compiling them together into a single unit that can be executed directly.

3. Programming in SEEK

SEEK programs are defined by one or more source files. Certain declarative information, described below, must appear early in the program source. These definitions are followed by module and rule definitions, in whatever order the programmer finds convenient.

Our SEEK implementation was built as an extension of the programming language C. In fact, SEEK programs are translated into equivalent C programs, hence the language can be used on any machine that supports a C compiler. All features of the C programming language are available to the programmer who uses SEEK, in addition to a number of significant extensions that support rule-based system features. This section concentrates on SEEK but also discusses particular features of C that are important to our work.

3.1. Encoding

The alphabet used for signal encoding must be declared to the SEEK compiler. This is done using an alphabet statement. For example, to indicate that the encoding for channel 2 uses an alphabet with symbols \{ F, BU, SU, SD, BD \}, one would write:

```
alphabet 2: F, BU, SU, SD, BD;
```

The symbols declared this way can be treated as constants by the program. Unique symbol names must be used for each channel even if the same encoding is used in more than one case. The only exception to this requirement is that the special symbol NOISE can be occur at any time on any channel if a significant noise burst is detected by the encoding expert. SEEK infers number of channels in the signal from the alphabet statements.

The encoding expert is called by the parser with a channel number as an argument. The encoded signal is returned as an array of symbols. The encoder is expected to compute a "reasonable" number of symbols on each invocation. In many systems, a specialized signal preprocessor computes the encoding offline and the encoding expert just reads information from a storage area. The encoding expert declares the symbol structure, which should include certain fields: e_type
(symbol type, from the alphabet), $s_{start}$ (start time), and $s_{end}$ (end time). Other information may be added to the symbol structure for use by the recognition algorithm (the rules).

### 3.2. Parts of a rule

The term *semantics* is used to denote "meaning," in contrast with *syntax* which denotes appearance. Each rule has a *syntactic* part and a *semantic procedure*. The syntactic part of a rule specifies a pattern that the parser employs to find matches between the rule and the encoded signal. The semantic procedure specifies some actions to be taken after a syntactic match is found. The appearance of a rule is as follows:

```
rule rname
{
  Syntactic part
  =>
  Semantic procedure
}
```

Here, the word *rule* indicates to *SEEK* that the following block of code defines a rule called *rname*. The rule itself is enclosed by braces: { and }. Inside, the syntactic part is separated from the semantic procedure definition by a "== >" delimiter.

#### 3.2.1. Syntactic part

The syntactic part of a rule is a set of labeled pattern specification statements. We will say that the label indicates the *pattern case*. Patterns are expressions that may contain:

- *Symbols*, declared in an *alphabet* statement. Symbols from different channels cannot appear in the same rule.
- The names of other rules, which need not be declared.
- The *or* operator: "|"
- Brackets: "[" and "]", denoting optional sections.
- Parentheses: "(" and ")".

---

1Time is represented using unsigned 32-bit integers.
Recalling the alphabet from section 2.2.1, the statement:

\[
\text{trapazoid: } (\text{BU} \mid (\text{SU} \text{ SU})) \mid \text{ F } \mid (\text{BD} \mid (\text{SD} \text{ SD}));
\]

declares a pattern case \textit{trapazoid} that would match symbolic encodings of trapazoids. It would also match encodings of other objects, but these could be distinguished by including appropriate semantic constraints in the rule. The pattern is understood to apply only to the channel encoded by the alphabet it uses. Examples include: “BU BD”, “SU SU BD” and “BU F SD SD”.

\textbf{Equivalence} statements are also included in the syntactic part of a rule. These are discussed in section 3.3.

\subsection*{3.2.2. Variables}

The semantic part of a rule resembles a procedure written in the \textit{C} language. It consists of declarations followed by statements. Special \textit{SEEK} extensions support the notion that knowledge can be \textit{private} to a rule-activation. In order to describe this storage class precisely, we must review the storage classes normally available in \textit{C}. In addition to having a type, a \textit{C} variable may be \textit{global} or \textit{local}, and \textit{static} or \textit{dynamic}. A \textit{global} variable can be accessed by any procedure in the program, while a \textit{local} (stack) variable is available only within the procedure or program block that declared it. Normally, local variables are allocated each time a block is entered and deallocated upon exiting; a \textit{static} local variable differs in that its value is retained across entries. \textit{Dynamic} storage is allocated by a program to create a new instance of some object from a storage \textit{heap}, then explicitly freed after use.

Recall that a \textit{SEEK} rule-activation manages private memory. This is accomplished by formalizing the handling of certain \textit{local dynamic} variables. When a variable is declared \textit{private} in a rule definition, \textit{SEEK} manages storage for that variable using the \textit{C} dynamic memory allocator. During creation of a new rule-activation, which occurs when a new instance of an object is learned, new memory is allocated for each private variable in the rule. Subsequently, when the rule-activation is used to interpret a segment of the signal, private variables are available for comparison with data derived from the signal and for update.
We can now understand a rule-activation as a binding of specific information (in the form of private variables) to the rule. Each time a rule is used to learn some object, a corresponding set of private variables is defined. We call this the private variable instantiation associated with the rule-activation. Private variables hold knowledge, while the rules provide structure and the algorithms to apply that knowledge.

Several mechanisms permit rule-activations to share information. To create a single database that can be accessed from all rule-activations, a global variable would be declared. As an example, this approach would be used by rule-activations to shape information about the shapes they have learned with the signal interpretation expert. The expert could use this information to identify noise-distorted instances of a shape using costly but highly robust analysis methods.

The export statement permits a rule-activation to make designated private variables visible to other rule-activations in the same rule-activation tree. Other rules in the tree use an import statement to obtain access to such variables. If two rules in a single tree export different local variables with the same name, a compile-time error message is produced. The export-import mechanism is useful when a rule measures some property of the waveform that is subsequently needed elsewhere, for example a revised estimate of waveform start and end times. Exported information can also be accessed by the output expert.

3.2.3. Semantic procedure

The semantic part of a rule is a procedure that defines the rule's behavior after a syntactic match is detected. The normal C assignment and control statements (if, then, else, for, while, do, switch, case, etc.) are available in SEEK. In addition, several special statements control the behavior of the rule during learning and subsequently, during scanning.

The Initially statement specifies an action to be taken only while creating a new rule activation. Normally, this statement is employed to initialize private variables, which are otherwise set to zero when a new instantiation is first created. Modification of private variables poses a difficult problem. Recall that a match is not considered successful until conflicts have been resolved. It follows that if a rule computes new values for its private variables while evaluating a segment of
the signal, the new values may be subsequently invalidated if the match is eventually rejected. For this reason, modifications of private variables are noted during the evaluation of a rule, but do not take effect until all conflicts are resolved.

While scanning, a rule estimates the probability that a match was found by comparing information in its private variable instantiation with a segment of the signal selected by the parser. This probability ranges from 0.0 (no match) to 1.0 (good match). To indicate the probability the rule executes one or more assert statement. During learning, assert statements can be combined with initially statements to enforce constraints on the objects the rule is able to learn. Thus, a rule to learn only tall, thin triangles might initially assert that the height of the object exceeds its width. To ensure that subsequent triangle instances match the prototype learned by a rule-activation, the rule would also assert that there is an approximate equality between the height of any candidate for a match and the height saved in its private variable instantiation.

Since rule-activations are often grouped into trees, it is generally necessary to compute an overall probability from the probabilities asserted by individual activations. This is done by taking the minimum\(^2\) of the asserted values. Normally, if the probability of a match drops below .7, the match is considered to have failed. The program can, however, vary this threshold.

SEEK provides a variety of facilities for access to the signal and manipulation of templates. The signal is available through a function call that locates a requested segment and returns a pointer to it:

\[
\text{ptr} = \$signal(\text{start, end});
\]

Subsequently, the point in channel chan at time rtm relative to the start time is accessed\(^3\) as a member of a two-dimensional array:

\[
\text{point} = \text{ptr}[\text{chan}][\text{rtm}];
\]

To make use of this facility, the rule must be able to determine where in the signal a tentative match was found (that is, why the rule was invoked). SEEK lets the rule determine which pattern

\(^2\)Other methods of combining probabilities might also be useful.

\(^3\)Some applications reduce storage requirements by compressing the signal. In such cases \$signal() returns a pointer into the compressed data.
case matched and the time at which the encoded symbols that matched the pattern were found. To determine the pattern case, a pcase statement is executed:

```haskell
pcase (casename, ...) 
{
  Statements
}
```

The statements are executed only if one of the listed cases was matched. Special functions are used to obtain information about the pattern. Specifically,

```haskell
$pattern[i]
```

refers to the i’th item in the pattern expression that matched. The pattern as a whole is:

```haskell
$pattern[*]
```

Functions that take $pattern as an argument include $stime, the start time for the first encoded symbol matched by this pattern, and $etime, end time. The function $symbol returns a pointer into the symbol array and can be used to obtain access to any other information passed out by the encoder in the symbol structure. Thus,

```haskell
$stime(pat)
```

is equivalent to

```haskell
$symbol(pat)->s_time
```

The function $count returns the number of symbols present in the matched segment.

### 3.3. Learning

In this section, the manner in which the interpretation expert controls the selection of an appropriate interpretation of a segment of the signal and adds it to the knowledge base is described. The behavior of a SEEK rule as it learns a new object and the nature of syntactic learning in SEEK are also discussed.

Assume that some segment of signal cannot be interpreted using the existing knowledge base. Thus, an encoding has been computed and either no matches were found when the patterns of the activated rules were used to parse the segment, or matches were found but the probabilities asserted by the rule-activations were low and hence all rule-activation trees that matched were


subsequently rejected. Under these circumstances, the interpretation expert is invoked with an indication of the segment in which parsing failed. Quick tests will often reveal noise contamination that was not detected by the encoding expert, in which case the parser is permitted to resume execution after the noise burst ends. Otherwise, the interpretation expert attempts to determine a correct interpretation of the signal. The expert may employ a variety of strategies to determine the correct interpretation, even consulting with a human supervisor if necessary. Once the interpretation is determined, a new rule-activation tree must be constructed and added to the knowledge base. To accomplish this, the parser searches for ways to parse the encoding of the segment containing the waveform using patterns defined by the entries in the rule-base. If no such parse is found, the waveform is one that cannot be learned, presumably because of an oversight during system design. Otherwise, one or more ways in which trees of rules could be used to interpret the segment will be discovered. One at a time, these trees are constructed by creating new rule-activations and linking them together. This is done by first creating a new private variable instantiation for each rule in the tree and then evaluating each rule-activation in initialization mode (the Initially statements are executed). Rules can place constraints on the objects that they can learn using Initial assertions (recall the example of "tall thin triangles" from section 3.2.3). If a tree is found for which these assertions are satisfied, the tree is added to the knowledge base and signal scanning is then resumed. We will say that such a tree is a prototype for the new shape. Trees are discarded if constraints are not satisfied (e.g., a low probability is initially asserted).

When a prototype rule-activation tree is added to the knowledge base, the parser extends the set of patterns that it searches for to include those defined by the prototype. Consider a two-channel ECG scanner that uses three rules to define QRS complexes. Say that rule qrs recognizes a QRS with appearance defined by rule chan1 in channel one and chan2 in channel two. Rule qrs will have a pattern of the form

\[
qrs: \text{chan1 chan2};
\]

whereas rules chan1 and chan2 have patterns based on the encoding alphabet for the appropriate channel. After learning a new object, a particular activation of rule qrs must be bound to
particular activations of rules \textit{chan1} and \textit{chan2}. For this purpose, new, unique rule names must be assigned to rule-activations when a prototype is constructed. Intuitively, the new rule-activation for \textit{qrs} is given a new name, \textit{qrs'}, with a modified pattern having the form:

\textit{qrs'}: \textit{chan1'} \textit{chan2'}

where \textit{chan1'} and \textit{chan2'} are also new names. Thus, \textit{qrs'} will never match an activation of the \textit{chan1} rule from a different rule-activation tree.

Recall that patterns can be split into multiple pattern cases. This feature is used to restrict the pattern part of a rule during learning. For example, a rule that matches either \textit{up} or \textit{down} triangles can be activated with a pattern that matches one or the other but not both. To do this, the \textit{up} and \textit{down} patterns are placed in different pattern cases. When \textit{SEEK} adds a prototype rule-activation tree to the knowledge base, only the pattern cases actually used during construct of that prototype are added to the set of patterns that the parser will search for. Other pattern cases are ignored. In some applications, it is desirable to indicate that several pattern cases should be activated simultaneously. The programmer does this with an \texttt{equivalence} statement:

\texttt{equivalence (casename, ...);} \\
All members of the equivalence class of a pattern case that occurs used in a prototype are activated simultaneously. (Case that are ill-defined because their patterns reference rules that were not activated during formation of the prototype are excluded when the parser updates its search-patterns.)

Sometimes it is necessary to partition pattern cases into those that can be used when constructing a prototype rule-activation tree and those that can only be activated by virtue of their membership in some equivalence class. In such cases, \texttt{initial assertions} are combined with a \texttt{pcase} statement to prevent the system from initializing prototype rule-activation trees that contain the excluded cases. Thus, a system can be specified that learns a shape in at least \textit{n} channels but subsequently permit matches with a subset of those channels. In a multi-channel system that permits recognition of waveforms in subsets of the total set of input channels, \textit{SEEK} finds the interpretation that includes information from the largest possible number of channels. In section
3.6 we describe a typical use of this feature.

3.4. Cache mechanism

When multiple interpretations conflict, several rule-activation trees may independently undertake identical computations on identical segments of the signal. Clearly, this is potentially inefficient. For example, if three shapes match a single segment of the signal syntactically, each may require the fourier transform of that segment during the ensuing activation-tree evaluations. *SEEK* provides an explicit cache function to store information temporarily. The results of costly computations can be saved in the cache and will be available for reuse at low cost. Information that is not reused is eventually deleted from the cache. In effect, the cache mechanism provides different rule activations with a way to share information, eliminating redundant computations.

Cache functions include: $\text{cache_add}(\text{key, information})$, to add information to the cache for retrieval using the named key, and $\text{cache_fetch}(\text{key})$, to obtain saved information that matches the key. The key specification is quite general and will not be described in detail in this paper. A typical request to add a vector containing fourier transform information computed from the signal at time $\text{time}$ might look like:

$$\text{cache_add}( \text{FFT, time, vector, sizeof(vector) } );$$

Here, FFT is assumed to be a predefined constant. *Sizeof* is a $C$ built-in function that returns the size of an object in bytes. While still in the cache, a pointer to the saved vector could be retrieved by the call:

$$\text{ptr} = \text{cache_fetch}( \text{FFT, time } );$$

A null pointer is returned if no information with key (FFT, time) can be found.

3.5. Conflict resolution

In section 2.4 (F) we described the conflict resolution expert. We now return to this expert to discuss its functions in greater detail. Recall that a *SEEK* system will discover every plausible signal interpretation consistent with its knowledge base. Circumstances inevitably arise in which several interpretations are discovered to conflict. The conflict resolution expert is invoked when-
ever a syntactic conflict is discovered between more than one rule-activation tree and is passed a pointer to a structure describing each match. The conflict resolution expert may immediately reject any or all of the interpretations, or can request that one or more of the trees be evaluated in detail. Subsequently, the probabilities computed during evaluation can be combined with acquired information ("experience") to resolve the conflict by rejecting interpretations or accepting the multiple match.

Conflict resolution plays a critical role in rule-based ECG analysis. Reasons for this and the algorithm used for conflict resolution in ECG analysis should clarify the function of the conflict resolution process. Recall that large T-waves can be separated from a QRS complex by essentially the same interval as can a PVC. In a SEEK program that has learned some PVC shape resembling the T-wave in both channels, one would expect T-waves to be occasionally detected as apparent instances of the PVC shape. In such a situation, the conflict resolution expert would normally be invoked. By examination of the signal or consultation with a human, the expert learns that this sort of conflict should be resolved by rejecting the PVC hypothesis if the interval between QRS and apparent PVC falls within certain bound (this would be based on observation of the normal interval between QRS offset and T-wave onset, which is shorter and more stable than the interval between QRS offset and PVC onset). Any repeated occurrence of the same conflict can be identified directly after parsing and corrected without actually evaluating the trees that correspond to the PVC hypothesis.

Note that SEEK addresses this potentially troublesome issue in an inexpensive but effective manner. Unlike the many ECG systems that use eye-closing (with its attendant risk of serious interpretation errors), SEEK systems detect every signal interpretation consistent with the knowledge base. For example, if artifact contaminates the segment of signal preceding some PVC shape, many systems would detect the artifact as a premature, abnormal QRS: a PVC. The (normal) QRS would be missed during eye-closing. A SEEK system might discover that the artifact can be interpreted as a possible PVC, but would also discover the normal QRS that follows it.

*The programmer specifies the minimum distance that can separate non-conflicting events.*
During conflict resolution, the conflict is easily resolved on the basis of simple rhythm criteria. Despite investigating extra signal interpretations, a SEEK system can be at least as fast as a less sophisticated program because it avoids investigating any signal interpretations that are clearly unreasonable: the amount of computation is determined by the plausibility of the interpretation being evaluated. Moreover, if an interpretation error is avoided the ultimate cost of the analysis may be reduced even if slightly more work is needed to reject the erroneous interpretation. In the majority of cases, a SEEK program rejects incorrect interpretations very early, perhaps during syntactic analysis – the cheapest operation performed during rule-based signal analysis.

3.6. Multi-channel signal analysis

Some of the most difficult issues in rule-based multi-channel signal processing arise from algorithms that tolerate occasional noise and dropouts on a subset of the channels (presuming that "critical" channels are unaffected). For example, figure 2 illustrates the use of a multi-channel rule to interpret a signal in which channel 4 is noise-contaminated and channel 6 exhibits low amplitude waveforms that cannot be detected. When rule-based systems are employed in such applications, explicit handling of channel loss must be provided by the rules. The intent of this section is to describe the SEEK features that assist developers of such rules. It should be stressed that this material is likely to evolve as we gain more experience with multi-channel analysis.

A rule-based system that does not employ the subset recognition mechanism described below will be totally unaffected by the special features outlined in this section. The section should therefore be regarded as an aside for those interested in the more difficult problems that arise when recognition of arbitrary subsets of a complex multi-channel knowledge structure is permitted.

The general strategy we have adopted permits "high-level" rules to indicate that "lower level" rule matches are optional. Ways to build patterns having this property have already been described. For example, if chan_i is a rule to recognize a waveform in channel i, a high-level "multi-channel" rule might use the following pattern:
threechan: chan_1 chan_2 chan_3;

The *threechan* pattern requires that a match be found in all three channels. To weaken this requirement one would add:

\[
\begin{align*}
\text{chan12: chan}_1 \text{ chan}_2; \\
\text{chan13: chan}_1 \text{ chan}_3; \\
\text{chan23: chan}_2 \text{ chan}_3; \\
\text{equivalence (threechan, chan}_12, \text{ chan}_13, \text{ chan}_23); 
\end{align*}
\]

Here, three pattern cases are introduced for recognition of subsets of the total channel set. The *equivalence* statement indicates that (if possible) all these patterns should be learned simultaneously. In fact, however, if one of the two-channel pattern cases is used during learning, the other cases would be meaningless because they each employ a rule that will not have been activated. As we indicated in section 3.3, such cases are not learned. It follows that in the example above, the equivalence statement has no effect unless the *threechan* pattern is originally used to detect the waveform being learned. To specify that only waveforms matching the *threechan* pattern can be learned but that 2-channel subsets can be used during subsequent recognition, we would include the following code in the semantic part of the rule:

\[
\begin{align*}
\text{pcase (chan12,chan13,chan23)} \\
\{ \\
\quad \text{Initially assert (0 /* Always fails */ );} \\
\}
\end{align*}
\]

The assertion is always false, hence pattern cases *chan12*, *chan13*, and *chan23* can never be learned directly. If the *threechan* case is learned these subset cases will also be activated. During scanning, a rule of this sort uses additional *pcase* statements to determine whether any dropout has occurred and then takes appropriate action.

Figure 3 shows a signal in which two waveform shapes are present, one corresponding to the normal QRS shape and the other to a PVC shape. Assume that these shapes are represented by two rule-activation trees based on a single set of rules. Further, assume that we wish to design a system that tolerates occasional loss of the signal in either channel. Using the pattern subset method outlined above, it is easy to specify this behavior syntactically. However, there are now a wide variety of possible interpretations of the PVC morphology shown in the figure. First, the event can be interpreted correctly using both channels of the PVC rule-activation tree.
Figure 2: Multi-channel scanning with signal loss in channels 4 and 6.

Figure 3: Using conflict resolution to distinguish PVC complexes from normal QRS complexes that are similar in one channel.

Alternately, either of the channels can be ignored, giving two single-channel cases of the PVC interpretation. Finally, the event can be interpreted as single-channel instance of the normal
QRS shape in which the second channel is distorted by noise. Thus, four different interpretations are consistent with our informal description of subset recognition in multi-channel rules.

The SEEK parser is implemented in a manner that simplifies this problem. Specifically, the parser will always find maximal syntactic matches. The problem in figure 3 is thus reduced to one of distinguishing between the two interpretations illustrated: the other interpretations arise from sub-maximal syntactic matches of the PVC rule-activation tree. A two-phase evaluation strategy is employed when evaluating a rule that supports subset recognition. Assume that a maximal match of a rule-activation tree of this sort has been evaluated, but was rejected because a subset of the single-channel rules employed failed to match (asserted a low probability). Normally, such a tree would be rejected. In a two-phase analysis, the tree-rejection procedure checks to see if some other pattern-case would have resulted in a sub-maximal syntactic match that omits the rules that failed. If so, the alternate case is evaluated (the next section discusses the problem of ambiguity, where more than one alternative satisfies this criteria). This strategy is computationally efficient because the parsing algorithm is essentially unchanged and low-level rules activations have already been evaluated: only the rule-activation using the new pattern case and any rule-activations that rely on information it exports need be reevaluated. Note that sub-maximal matches are only used to exclude rules that failed to match. A subset match would never exclude rules that asserted a high probability of match except as a side-effect of excluding some other rule that failed.

Conflict resolution becomes more complicated when two-phase evaluation is employed. For example, a conflict resolution expert would have to learn to resolve the conflict shown in figure 3 in favor of the PVC interpretation. Moreover, the two-phase evaluation strategy implies that "rejection" of an interpretation can have the side-effect of causing an alternative interpretation to be discovered. Conflict resolution experts that can handle this possibility are only slightly more difficult to design than those used in an environment that does not include multi-channel subset recognition. To simplify the development of such algorithms, conflict resolution experts that implement basic multi-channel analysis strategies have been included in a "library" of conflict resolution experts for use as starting points when building complex multi-channel pattern-
recognition systems.

The two-phase evaluation strategy is not entirely satisfactory because it constrains the design of a multi-channel system in the following manner. If an interpretation requires exclusion of a channel, rejection of that channel must take place in the rule associated with the channel and not in a higher level rule that operates on several channels. Otherwise, subsets omitting that channel will inevitably omit other channels too. This restricts the manner in which computations can be structured, making it more difficult (not impossible) to postpone expensive computations until the final phases of evaluation. Nonetheless, this is primarily a "stylistic" objection: two-phase evaluation successfully addresses a difficult issue and it appears that most analysis algorithms can be formulated in terms of this approach.

3.7. Parsing the encoded signal

A discussion of the function of the parsing expert would draw extensively on well known results from formal language theory. For this reason, although the selection of efficient parsing algorithms poses significant problems, discussion of the parser is omitted. The interested reader is referred to [18] [21], where algorithms for parsing ambiguous languages are described. Observe that any single-channel parser can easily be extended to a multi-channel environment if the maximum "skew" between two channels is limited, as is the case in a SEEK system.

3.8. Ambiguity

The syntactic analysis undertaken by the SEEK parser involves recognition of a highly ambiguous language: each rule-activation tree can conflict syntactically with many others. Despite supporting this form of implicit ambiguity, SEEK presently does not support systems in which a single rule-activation can match a single segment of the encoded signal in more than one way. Otherwise, rule-activation trees could conflict with themselves. An implication is that rules should define pattern equivalence-classes so that only one case can match if a two-phase evaluation is undertaken. Ambiguity is detectable under most conditions and appropriate warning messages are produced. Eventually, explicit resolution of ambiguities will be added to the language.
4. Detailed example

Consider the partial SEEK program that appears in figure 4. This section discusses the different rules used in the example and their behavior when the program is run.

Recall that a multichannel recognizer coded in SEEK must declare the maximum skew that can separate waveforms in each channel. In the example, this is done using three define statements. Define is part of a C macro facility that requests in-line substitution for the defined object.

```c
/*
 * A SEEK program to recognize 2-channel QRS complexes
 * It assumes an encoding of the signal using triangles.
 */
#define TLEN 20  /* Points/template */
#define EXTRA 10 /* Larger window for centroid */
#define MS(msecs) (msecs/4) /* Convert msecs to points */
#define MAXSKEW MS(100) /* Maximum skew, for parser */

/* Declare the encoding alphabet */
alphabet 1: UP1 DOWN1;
alphabet 2: UP2 DOWN2;

/* The QRS rule matches dual channel waveforms */
rule qrs
{
    dual: chan1 chan2;
    private int qskew, rtime, templates[2][TLEN];
    import int rtime1, rtime2;
    export rtime;
    int skew, temp[TLEN], qrsrime, chan;
    skew = rtime2-rtime1;
    initially qskew = skew;
    assert (qskew ~ skew);
    qskew = (qskew+4 + skew) / 5;

    /* Make templates for channels 1 and 2 */
    qrsrime = (rtime1+rtime2)/2;
    for(chan = 1; chan < 2; chan++)
    {
        make_template(temp, skew, temp1);
        Initially templates[chan] = temp;
        assert (corr(templates[chan], temp) > .8);
        update(templates[chan], temp);
    }
    rtime = rtime1 + skew/2;
}

/* Rule to learn and recognize a waveform in channel */
rule chan
{
    /* Match a UP or DOWN triangle */
    oneu: UP1;
    oned: DOWN1;
    /* Match an UP-DOWN or DOWN-UP sequence */
    twou: UP1 DOWN1;
    twod: DOWN1 UP1;
}

private int areal, areas, rtime1;
export rtime1;
int area;

area = $symbol(pattern[0])->a_area;
initially areal = area;
assert (areal ~ area);
areal = (areal+4 + area)/5;

pcase (twou, twod)
{
    area = $symbol(pattern[1])->a_area;
    initially areas = area;
    assert (areas ~ area);
    areas = (areas+4 + area)/5;
}

rtime1 = $symbol(pattern[0])->as_rtime;

/* C subroutine to make a template */
make_template(time, time1, chan)
int *temp, time, chan;
{
    register int i, offset, spt[ || ];
    spt = $signal(time-TLEN/2-EXTRA, time+TLEN/2+EXTRA);
    offset = centroid(chan, spt);
    for (i = 0; i < TLEN; i++)
        temp[i] = spt[chan][i+offset];
}
```

Figure 4: SEEK program for 2-channel QRS recognition
using the indicated definition. Thus, when "TLEN" is used in the program, the constant "20" is substituted. Similarly, the example defines a function, MS(x), which converts from milliseconds to data points, assuming a sampling rate of one point every four milliseconds. A maximum skew of 100 milliseconds is then declared.

The next lines of the program declare that the encoder uses an alphabet with symbols *UP* and *DOWN* (qualified with channel numbers for uniqueness). With this in mind, consider the *chan1* rule, which learns and subsequently recognizes single-channel waveforms. An assumption is made that each QRS encodes into a single triangle or a sequence of two oppositely directly triangles (a larger-scale system might employ more sophisticated patterns). Each pattern is given a case name: *oneu*, *oned*, *twou*, and *twod*. Next, assume that this rule is taught to recognize the waveform shown in Fig. 1, channel one. The interpretation expert would first determine that the QRS has been encoded using the sequence *UP1-DOWN1*. The *chan1* rule would then be activated using pattern case *twou*. New memory would be allocated for the private variables *area1*, *area2*, and *rttime1*.

A temporary local variable, *area*, is used for convenience. It holds the area of the first triangle:

```c
area = $symbol($pattern[0])->s_area;
```

It should be noted that this statement assumes that the symbol structure included a field *s_area*. Since the rule is being activated, the *initially* statement is executed and the area is saved in the private variable *area1*. The subsequent *assertion* checks for approximate equality between *area1* and *area*, a vacuous test during initialization. Similarly, the statement that updates the value of *area1* is executed without effect during initialization. Recall that pattern case *twou* was used for the match, hence an identical computation is used to initialize *area2* from the area of the second symbol in the encoded QRS. Finally, a private variable *rttime1* is set to the start time of the first symbol that matched. This information is exported for use by the *qrs* rule (primarily to illustrate the export mechanism, since the *qrs* rule could have determined the start time directly using a similar computation). A more sophisticated *delineation refinement* computation would be
employed to identify waveform onsets in a full scale system [1] [2].

After the rule-activation has been created, recognition of chan1 instances proceeds as follows. The parser discovers a match with the activated twou pattern and executes the rule to interpret the corresponding segment of signal. The initially statements are now ignored and the assert statements therefore test for approximate equality with the saved area1 and area2 values. Updates to area1 and area2 are computed and recorded, but do not become permanent at this time. A four-point moving average is used to compute the updated values.

The chan1 and chan2 rules are assumed to be identical.

Now consider the qrs rule. During learning, this rule measures the skew in milliseconds between the waveforms in each channel. It also saves templates from the waveform in each channel. During normal scanning the qrs rule activation is invoked only after the corresponding chan1 and chan2 activations have been evaluated and, through their assert statements, confirmed the apparent match. The qrs rule computes the skew between channels and checks for approximate equality with the average skew. If the test succeeds, a template correlation is used to compare the shape "known" to the activation with the instance that has been detected. As in the case of the chan rules, updates to private variables are computed but not saved permanently. Note that it would probably be better to implement the template comparisons in the chan1 and chan2 rules, because subset recognition is otherwise impossible. The algorithm used in the example was, however, formulated to illustrate SEEK and the export-import mechanism, and not for maximum generality.

Using a conflict resolution expert, not shown, the SEEK program determines the correct interpretation of each segment of the signal. Each time an interpretation is found, the corresponding rule-activation tree (or trees) are updated by changing private variables using the values computed during rule evaluation. At this time, the output expert is also invoked. This procedure (not shown) can access the three variables rtime, rtime1, and rtime2.

Finally, consider the procedure make_template. This is a normal C procedure that uses SEEK extensions to form a template based on a window surrounding the centroid of a detected
waveform. Other procedures used by the program for centroid computations, template correlation, and to update templates are not shown. The C language permits the programmer to specify that certain variables and pointers should be placed in hardware registers for extra performance. Here, this feature is used to optimize a loop in order to minimize the cost of template formation.

5. Status of the project

This section gives specific results from our ECG research and outlines areas for future exploration.

5.1. ECG Analysis

A single-channel ECG analysis system was developed using SEEK and was evaluated using signals that provoke systematic errors when conventional analysis algorithms are used. The evaluation was done by analyzing 15 "difficult" 20-minute ECG recordings using Columbia-V, a scanning system that we developed for use by researchers at Columbia University. Columbia-V does not use rule-based learning algorithms and is prone to systematic error under some conditions. Performance of the prototype SEEK system was then compared with that of the Columbia system. Detailed results appear in [1] and [2].

Our tests were designed to measure system performance during detailed waveform analysis. In particular, the rates of minor waveform detection errors and of systematic errors were measured. Our rule-based system achieved a 53% reduction in minor detection errors and nearly complete elimination of systematic error in our test data. Speed of the rule-based algorithm was limited by the number of discrete fourier transforms and correlations performed, since these were needed when comparing waveforms. We found that an average of 1.8 transforms were computed and that an average of 3.7 template comparisons were performed per detected complex. In a dual-channel system, similar rates would be expected. Columbia-V uses a different shape classification method that performs a fixed (but larger) amount of work on each complex. This suggests that a fast ECG analysis system can be constructed using our methods. It should be stressed that this evaluation was quite preliminary and may not reflect the behavior that would be expected from a fully optimized, dual-channel system. However, these results were very
encouraging. We are now designing a two-channel scanner that will employ dual-channel analysis algorithms.

5.2. Other signal processing applications

SEEK has clear application to many of the other signal processing problems that have been studied in the literature. Carotid pressure waves, cardiac mapping systems, nerve discharge waveforms, and EOG signals all exhibit repetitious structure for which learning algorithms might yield improved analysis performance. SEEK may be particularly useful in the analysis of signals coming from large numbers of channels, such as the signals collected in cardiac surface mapping systems.

5.3. Image processing problems

The adaptation of our work to image processing poses especially interesting problems. Much work in image processing has taken a structural approach, utilizing hierarchical knowledge structures similar to those represented in our rule-activation trees [22]. Adaptation of SEEK to image processing would require the definition of easily computed primitive images for use during an encoding operation. It would also be necessary to devise a 2-dimensional pattern language. A specific medical image processing problem to which SEEK might be adapted is the automatic interpretation of ultrasound images.

5.4. Implementation status

A SEEK compiler for the UNIX operating system has been implemented on our VAX 780 and will be moved to a SUN M68010-based microcomputer that also runs UNIX. The compiler and associated libraries of support procedures will be made available to other sites once a sufficiently reliable version is completed. SEEK compilation is very fast and algorithm development is therefore straightforward. Integration of the language with UNIX debugging and program optimization facilities should be practical, and will be attempted in the future. The compiler is relatively simple although the current parsing algorithms may not be optimal. We plan further investigation of this issue.
6. Summary and conclusions

This paper described our rule-based approach to solving learning problems in multi-channel signal processing. Our discussion focused on a new language, SEEK, which simplifies the problem of rule-based program development, making the methodology accessible to researchers investigating related problems in other areas. SEEK is suitable for use in a wide variety of multi-channel signal processing problems, although the handling of ambiguous interpretations and analysis of subsets of channels needs additional study. Research on difficult signal interpretation problems may eventually lead to a better theoretical understanding of the nature of multi-channel information. Our rule-based ECG analysis algorithms demonstrate the significant potential of the rule-based approach, particularly in multi-channel signal processing.

7. Acknowledgement

Many revisions to an early draft of this paper were suggested by F. Schneider, whose assistance is gratefully acknowledged.
References


1980.
