Final Report
"Interdisciplinary Investigations in Support of Project DI-MOD
Research Grant: NAG 2-670

Prepared for
NASA Ames Research Center
Mail Stop 242-4
Moffett Field, CA 94035

b y

Principal Investigator: Dr. Scott A. Starks

Department of Electrical and Computer Engineering
University of Texas at El Paso
El Paso, Texas 79968-0523

April 11, 1996

1. Background

A research grant was awarded to The University of Texas at El Paso (UTEP) by the NASA Ames Research Center for the purpose of enlisting the services of the University in the investigation of approaches to assist in the development of a ground station architecture to support remote sensing and geographical information system applications. Various concepts from time series analysis were to be used as the basis for the development of algorithms to assist in the analysis and interpretation of remote sensed imagery. Additionally, research was conducted toward the development of a software architecture to support processing tasks associated with databases housing a variety of data . This report presents an algorithmic approach which was developed which provides for the automation of the state monitoring process. The power specta of time series of sensor data provide the data which we process to perform monitoring.

2. Trend Monitoring via Fractal Analysis of Power Spectra

As systems become more complex, state monitoring and trend detection have become increasingly difficult tasks. With the advent of new technologies, processes requiring a considerable level of human interaction are being replaced with others employing higher levels of automation. Various types of systems encountered in nature, such as an eco-system, presents challenges to those interested in studying the level of change (e.g., environmental degradation) being encountered. Both state monitoring and trend detection are critical in the developmen of effective methods for determining the state of health of complex systems. Trend detection is considered to be a more difficult task than state monitoring due to the fact that one must incorate some sort of model against which to base the future behavior of the system given previous observations.

In this section, we present an approach to trend detection that is based upon the fractal analysis of power spectrum estimates. Given that timeliness is an important consideration in the evaluation of any trend detection scheme, we were led to such an approach. Primarily because of its ability to condense large volumes of data into a few number of parameters, fractal analysis appears to be a reasonable approach Before we illustrate our approach, we will introduce the mathematical basis of our approach. Next, we present an algorithm which enables the estimation of parameters of the fractal model in a very time efficient manner. Lastly, we discuss the application of this approach to trend detection within the context of system monitoring

As with any mathematical representation in nature, fractal models can be treated as abstractions. An excellent review of fractals can be found in [S86]. Fractals can be categorized as being either "exact" or "statistical." Just as circles and squares are geometrically regular, exact fractals exhibit regular properties. Because exact fractals rarely appear in nature, we shall concentrate on statistical fractals in this study and illustrate how they can be used to model physical processes.

Statistical fractal models can be constructed to depict patterns occurring in imagery such as those relating to land use, water resources, vegetation, mountainous terrain and human population. Parameters drawn from fractal geometry can be used as features which can be useful in image analysis, segmentation, characterization and interpretation. In particular, the level of complexity present in patterns can be compactly described using what is known as fractal dimensionality. Dimensions are essential in both Euclidean and fractal geometry for representing characteristics such as area, density and perimeter length.

Fractals have dimensions, just as points, lines, planes and solids have topological dimensions. Any of these aforementioned geometrical objects may be thought of as sets of points located in a space of a particular dimension. As an

example, a single point is a geometric object with a dimensionality d=o. A line is a collection of points arranged such that its resulting dimensionality is 1. In turn, two lines may be used to define a two-dimensional plane, etc.

Similarly, fractals may be treated as subsets of higher dimensional spaces. Unlike Euclidean geometry, fractal dimensionality can be and most often is a non-integer quantity. Fractal dimensionality involves the relationship between a quantity Q, and a length scale L, over which we measure Q.

Let us consider an example taken from [O92]. Suppose we wish to locate fast food restaurants on the main street that connects the campus of the university to downtown. We want to locate these restaurants in such a way the distance from every point on the street (which may be represented by a straight line of length L) to one of the restaurants is e or smaller. If we place restaurants in such a way that their number N(e) is minimized while maintaining the constraint that no point on the street be no further from the nearest restaurant than e then, we are led to the expression

$$N(e) \approx L / (2e) \tag{1}$$

Now, let us consider the case where we wish to cover a regular two-dimensional geometric region of area A. Once again we wish that no point of the region be at a distance more than e from the nearest restaurant. We can construct a service zone of area = (2e)2 about each restaurant site. Then the smallest number of restaurants needed to cover the region of area A can be approximated by

$$N(e) \approx A / (4e2). \tag{2}$$

Finally, let us consider the case where we need to cover a three dimensional region, e.g. a mall of volume V. One again, we wish to locate restaurants in a manner that each point in the volume is at a distance no greater than e from the nearest restaurant. We may construct a zone of volume (2e)3 about each restaurant site. In doing so, we may approximate the number of restaurants needed to fill the volume while at the same time satisfying our constraint as

$$N(e) \approx V / (8e3). \tag{3}$$

It is reasonable to give the following definition of dimensionality. If for some set S, the corresponding number N(e) satisfies the following condition

$$N(e) \approx C / e^{\partial}. \tag{4}$$

for some positive numbers C and a then we call ∂ the fractal dimensionality of the set S. This definition of fractal dimensionality may or may not be an integer. In representing irregular curves or regions lying on a plane, it is reasonable to expect the fractal dimensionality to lie between 1 and 2. It can be gleaned from the example that the fractal dimensionality represents the way in which a quantity Q varies with scale.It can be shown ∂ corresponds to dimensionality in the regular sense. As a consequence, a fractal may be thought of as a subset obtained by extracting some portion of the space within which it resides.

Suppose that we are presented with a real signal, x(t). We may create a time series of data by taking samples of x(t) at equally spaced intervals of time. Let us denote this time series as before. Suppose further that we have available the power spectrum of the signal, i.e. the square of the signals Fourier transform. We wish to model an approximation to the spectrum by means of a function

$$S(w) \approx Aw^{-\partial} \tag{5}$$

where we see there are just two parameters of the model. Depending upon the nature of the spectrum, the values of the parameters will vary in such as way that the model fits the spectrum accordinging to some optimality criteria such as least squares. As an example, the power spectrum of a Brownian motion sequence is described by the equation (5). Trajectories of processes with the spectrum of the form turn out to be fractals. Therefore, we chose to approximate the power spectrum of random time series using the form given in (5)

If we measure the values of the spectrum for at n values of frequency then we are left with solve the overdetermined system of equations.

$$\begin{aligned} S(w_1) &\approx Aw_1^{-\partial} \\ S(w_2) &\approx Aw_2^{-\partial} \\ &\cdots \\ S(w_n) &\approx Aw_n^{-\partial} \end{aligned} \tag{6}$$

These equations are nonlinear in the parameters A and ∂. By applying the logarithm to both sides of the equations of (16) and rearranging terms, we obtain the set of equations

$$\begin{aligned} \ln\ S(w_1) &\approx \ln(A) - \partial\ \ln(w_1) \\ \ln\ S(w_2) &\approx \ln(A) - \partial\ \ln(w_2) \\ &\cdots \\ \ln\ S(w_n) &\approx \ln(A) - \partial\ \ln(w_n) \end{aligned} \tag{7}$$

Allowing the substitutions

$$y_i = \ln\ S(w_1) \tag{8}$$

and

$$x_i = \ln(w_1) \tag{9}$$

yields a set of equations in a more familiar form

$$y_i = a\ x_i + b \qquad \text{for } i = 1, 2, ..., n \tag{10}$$

where

$$a = -\partial \tag{11}$$

and

$$b = \ln(A) \tag{12}$$

Due to the convenient form of the equations given in (10), any one of a number of well-known least squares methods can be used for solution of a and b. The transformation equations (11) and (12) can then be applied to a and b to yield the fractal dimensionality and amplitude. A computational approach is presented in [S93].

The earth and its ecology comprise one of the most complex systems known to man. Fractal analysis can play in the interpretation of remotely sensed imagery which provides information about the state and trend of geographical areas. Landscape structure affects many physical processes. For example, the spread of a disturbance such as a forest fire and the landscape pattern on which it acts are highly related. Likewise, landscape patterns influence the movement of resources, humans and wildlife. In making forecasts, landscape ecologists face the task of quantifying patterns of great temporal and spatial complexity. Fractals can be viewed as a tool for quantifying landscape patterns and can be used in a number of situations for modeling landscape processes. It is clear that the current state of technology provides us the ability to acquire and store huge amounts of data. However, it remains a problem to access and analyze the data to produce information content which can, in turn, be used by scientists and policy makers.

Fractals can be used in analyzing remotely sensed imagery such as the type stored in a regional database. Previously, we have presented an approach to the characterization of the power spectra of time series using merely two parameters from fractal analysis. An approach has been presented for the efficient computation of these parameters thus allowing such an analysis to be possible in real time. We are currently investigating the application of the algorithm as a means of region segmentation in remotely sensed imagery. This application involves analyzing time series resulting from the visible and infrared reflectance of vegetation ground covers obtained from remote sensing instruments. As we know well, vegetation development is highly seasonal with different species developing at different times of the year and with different growing seasons. We are able to merge areas on the basis of the behavior of their time series.

Fractal dimensionality is a parameter that can be easily computed for areas of interest in a given geographical setting. We plan to utilize fractal dimensionality as a basis for image segmentation for remotely sensed images obtained from NASA as well as other sources. In particular, we are keenly interested in aggregating regions which appear to have the same structure as determined by fractal dimensionality so that we can make intelligent comparisons about regions supporting similar land cover and land use.

We plan to apply fractal analysis to the power spectra obtained from time series observations of physical quantities such as the greenness index which is used to classify land cover. We believe that areas undergoing similar patterns of seasonal change will yield similar values for the corresponding fractal dimensionality. In addition, features indicative of dramatic global change will also arise from studies involving the fractal analysis of time series.

In [M91] a number of potential applications of fractal analysis to quantitative landscape ecology are presented. It is known than fires spread across landscapes along a connected network of fuel. Rarely could fuels in a forest or grassland be envisioned as uniformly occupying the plane. Instead , a subset of the plane may be rich in fuels and may show consistent changes in mass or density with scale. Such a distribution of fuel may be characterized using a fractal model

The velocity of a moving fire front can be modeled as

$$v=(p-p_c)\ (D_{min}-1)$$

where p is the percentage of landscape occupied by fuel, p_c is the critical probability above which the fuel forms a contiguous cluster across the landscape, and D_{min} is the fractal dimension of the path between any two points residing in the contiguous cluster. Other application areas include those in the areas of modeling the ability of species within landscape structure and the modeling of variables such

as soil moisture as a function of spatial location or time. Fractal models of these sorts provide a basis for investigation of the impacts of global change on population distributions, land use and agricultural yields.

3. A Software Architecture for the Analysis of Geographic and Remotely Sensed Data

Many federal agencies acquire, store, and process geographical data (which is typically acquired through ground studies) and image processing data (which is typically acquired through remote sensing). A significant amount of remotely sensed data is acquired through satellite observations. Software systems exist which provide for spatial and temporal query access to information about various regions of the earth. For example, from a low resolution map of the U.S. a user might outline an area such as Texas via a mouse and quickly view a map with much greater detail about Texas. In fact, the user might select from a number of possible views of Texas, including views which show roadways; vegetation; bodies of water and rivers; cities and towns; or any combination of these features and others not mentioned. From this standpoint the user could select a city in order to gather even more detailed information which might include streets, buildings, vegetation, fire hydrants, utility information (both above and below ground), population data, weather data, etc. Some of this data might come from remote sensing (for example buildings and streets could be obtained from visible data and vegetation from near-infrared data) and some data might be obtained from ground studies including the population and weather data. The user might also be able to obtain this same data along a temporal dimension. The remote sensing data is typically stored in a manner such that it is called *image data* or *raster data,* while the ground study data is called *vector data.* The vector data is typically associated with some *spatial key* and, from the database standpoint, might best be viewed as a tuple of data associated with some location.

The ability to view information as was just described requires a sophisticated software architecture (we will ignore the obvious hardware requirements). Figure 1 provides an overview of this architecture. The data acquired from remote sensing and ground studies must be preprocessed in order to make it suitable for display. Scientists develop exploratory programs which process the raw data, extracting those subsets of data which are of interest in some problem analysis. Image processing software is employed by analysts in order to develop displays of images which contain as much information content as possible. Once the data and images are of a sufficient quality they migrate into databases to be accessed, usually over networked systems, and displayed using a Geographical Information System (GIS). GIS systems typically provide access to all vector data and to some subset of image data. Some image processing information, however, cannot be combined with related GIS information. Clearly, the software architecture to support the analysis of earth data involves the use of a wide range of sophisticated software tools and exploratory programming languages.

There are two major problems with the current software architecture. The first problem is that it takes a significant effort to do the exploratory programming to extract data of interest from the raw data database. Furthermore, once the data has been processed and is residing in the processed data database, it takes too long to do further processing in a database host language or even in a procedural version of *SQL.* The second problem is that it is difficult to know how the high level tools will interact. In other words, there is no known formal specification language available to describe the functionality of the various tools which make up the architecture. If there were such a language and, for example, both the geographical information system and the database management system were specified in this language, the combined behavior of the two specified systems could be studied prior to attempts to

put the systems together.

4. The Need for Different Programming Languages

It is estimated that less than 3% of the Satellite Data available has ever been seen. Less than 1% has been analyzed. In other words, we have the technical know-how to acquire and store the satellite data but we have difficulty determining the information content contained therein. Every year that passes adds more data to the existing unanalyzed data sets. The central problem has to do with programming. It seems that much of the exploratory programming to analyze the raw data sets is done in languages like FORTRAN or C. The resulting programming effort costs too much money and takes too much time for it to keep up with the growth in new data.

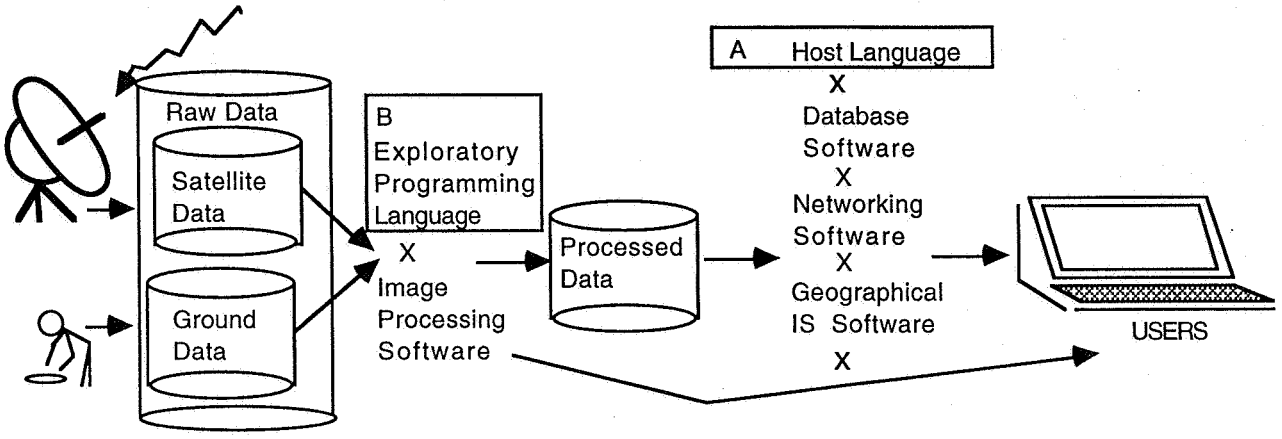

Figure 1. Software Architecture for Geographical Data Analysis.

SequenceL (formerly called BagL) is a language designed to experiment with a strategy for problem solving wherein one solves a problem by describing data structures strictly in terms of their form and content, rather than also having to describe algorithms to produce and/or process the data structures. The *SequenceL* work has led to the identification of constructs for describing data structures. Our view of a data structure is meant to include traditional data structures, databases, screen displays, reports, etc. The primitive data structure of *SequenceL* is the ***sequence***.

Sequences in *SequenceL* are collections of elements wherein each element may occur more than one time (as opposed to a mathematical *set*) and where each occurrence of an element possesses an ordinal position (as opposed to a *bag* or *multiset*). A ***sequence*** may be singleton (e.g., [99]), or nonsingleton (e.g., [[1],[2],[3]] or [[[1],[2],[3]],[[10],[20],[30]]). Complex structures of ***sequences*** containing ***sequences*** can be described. Like the list of LISP and the array of APL and J, the ***sequence*** of *SequenceL* can be used to build any data structure. We call the various data structures that can be composed using sequences, **nonscalars**.

It is believed that processing geographical data utilizing nonscalar processing constructs will improve the productivity in the exploratory and database programming efforts depicted in Figure 1, boxes A and B. This is our niche. Nonscalar processing constructs do not require the explicit use of iterative, recursive, or I/O constructs. Furthermore, early analysis of *SequenceL* seems to indicate that the nonscalar constructs provide a uniform abstraction for data definition, query, host language, and integrity constraint processing. Much effort is needed, however, to develop a robust, easy-to-use, production-quality language based solely upon these constructs. The production quality version of such a language must enhance our ability to develop large programs, particularly the types of

programs which comprise the result of exploratory programming efforts.

4.1 Nonscalar Primitives

A computational step, in *SequenceL,* involves the evaluation of a guarded command, which is applied to an input ***sequence,*** and which yields an output ***sequence.*** Functions contain the guarded commands of a *SequenceL* program. The inputs to a computational step are obtained in one of two ways: (a). through an **eventive** construct, wherein a function consumes its input from a universe of named sequences and produces, as its output, a set of named sequences (in the universe); or (b). through normal functional parameter passing, which is similar to the parameter passing in languages like LISP.

A total computation, in *SequenceL,* yields a partially ordered set of named ***sequences.*** A command, in a *SequenceL* guarded command, falls into one of three categories: **generative, regular,** or **irregular.** These three categories, combined with the **eventive** category, comprise the four constructs for nonscalar processing, and are used in place of algorithms.

There are two major concerns which arise when processing nonscalars: the selection of items to participate in the processing and whether processing reduces or expands input when producing output.

In nonscalar processing, a nonscalar (i.e., the *DOMAIN* column in figure 2) serves as the basis for a new nonscalar (i.e., the *RANGE* column in figure 2). A given construct acts in a manner to produce a *range* nonscalar based upon the *domain* nonscalar. All elements of a domain may participate in the computation needed to produce the range. The ***regular*** and ***generative*** constructs use all elements of a domain ***sequence*** in order to produce a range ***sequence.*** For example, when one computes the *sum* of a set of elements, the addition operator is being applied, in a **regular** fashion, to *all* elements of the set (i.e., the addition operator is acting as an aggregate operator in this case). Alternatively, some subset of the elements in the domain ***sequence*** are used to produce the range sequence. The ***irregular*** construct selects items to participate based upon position or value, while the ***eventive*** selects elements based upon availability and (optionally) upon additional conditions such as position or value of the available domain element.

In producing a range sequence, nonscalar constructs either reduce or expand the domain ***sequence.*** Reduction and expansion can occur with respect to the cardinality or dimensionality of the domain sequence.[1] Expansion means that the range ***sequence*** is larger, in dimension or cardinality, than the domain ***sequence*** upon which it is based. Reduction means that the range ***sequence*** is smaller, in dimension or cardinality, than the domain ***sequence*** upon which it is based. Both the reduction and the expansion may sometimes leave the range ***sequence*** equal, in dimension and/or cardinality, to the domain ***sequence*** .

In both the regular and irregular processing of nonscalars, one begins with a nonscalar and (typically) *reduces in the dimension* of the nonscalar (in the case of regular processing) or *reduces in cardinality* (in the case of irregular processing). The ***generative*** construct expands results.

	DOMAIN	RANGE	
	Selection	Dimension	Cardinality
Regular	All	Reduces	
Irregular	Some based on selection		Reduces
Generative	All	Expands	Expands
Eventive	Some based on availability	Reduce/Expand	Reduce/Expand

Figure 2. A Summary of Nonscalar Constructs.

4.2 Examples of Nonscalar Primitives in *SequenceL*.

The ***regular*** form of processing is used when an operation is to be applied in a uniform manner to *all* of the elements of the nonscalar. For example, when one computes the *sum* of a set of elements, the addition operator is being applied to *all* elements of the set (i.e., the addition operator is acting as an aggregate operator in this case). This construct typically reduces a structure in terms of its dimension. The example below reduces from a two dimensional sequence into a one dimensional:

**([[1],[2],[3],[4]]) » 2 [(1*(2*(3*4)))] » » [24]*

In other cases, one may wish to apply an operator to *some* of the elements of the nonscalar. The selection of elements may be based upon *value* or *position*. *SequenceL* possesses a construct for this ***irregular*** form of processing data. The ***irregular*** construct requires the ability to select items conditionally based upon a *map* or subscript. In *SequenceL* this construct requires a guarded command structure. The noniterative form of the guarded command provides the basic form of the body of a *SequenceL* function. For example, suppose one wished to select all odd values in some sequence of integers, *int*. To do so, one would need to provide a function:

```
Odd_Integers(Domain(Int),Range()) =
      [  [Int(i)]    when ≠(mod(Int(i),2),0)
       []            otherwise   ]
```

This function effectively states that one wishes to return each *ith* value of *Int* when that *ith* value is odd (i.e., when the *ith* value *mod* 2 is not equal to zero). *Odd_Integers* has no *range* arguments because it is intended solely for use as a helper function. That is, it is to be invoked by some other *SequenceL* function which will supply its argument *Int* and accept its result. Suppose one wished to sum all odd integers in *Int*. To do so, one would compose functions:

```
Sum_of_Odds(Domain(Int),Range(Sum)) =
      [  +(Odd_Integers(Int))    ]
```

This function is considered to be a nonhelper function.

A *SequenceL* program consists solely of a set of *SequenceL* functions which are applied to a *SequenceL* **Universe**, *U*. The **Universe** is where the user may pair *SequenceL* variable names with ***sequences*** of values via a text editor. Like GAMMA [B93, HLS92] (and similar to the tuple-space of **Linda** [G85]), when a *SequenceL* program executes it modifies the **Universe** to which it is applied. If each variable appearing in a function's *Domain* is paired with a ***sequence*** in *U* , the function is enabled for execution and is termed a **nonhelper function**. Obviously, several

functions may be thus enabled. These functions execute concurrently. When a nonhelper function executes, it consumes all variables in its domain. When a nonhelper function completes execution, the result(s) of the function is(are) paired with the function's range variable(s) and added (or produced) in the **Universe**. It is via the **Universe** that a user provides inputs and obtains outputs. There are no explicit constructs for I/O in *SequenceL*.

The interaction between a *SequenceL* function and its **Universe**, in terms of production and consumption, provides for the ***eventive processing of a nonscalar structure.*** In eventive processing, one processes a nonscalar structure whose elements are not necessarily available all at once. The arrival of an element is an event to which a function must respond.

Assume program Π is applied to universe *U*, where Π contains the functions *Odd_Integers* and *Sum_ of_Odds,* as introduced above, and *U = { <Int, [[6],[7],[8],[3]]> }.* Based upon the *eventive construct,* only *Sum_ of_Odds* is enabled for execution. It consumes its arguments from *U:*

U = { }

and immediately invokes *Odd_Integers* which is applied to each value of *Int*. *Odd_Integers* returns only those values which meet the *≠(mod(Int(i),2),0)* condition. Since, empty sequences *[]* are null values they disappear in the result of *Odd_Integers* which is returned to *Sum_of_Odds:*

Sum_of_Odds(Domain(Int),Range(Sum)) = [+([[7],[3]])]

Notice that *Odd_Integers,* which is an example of irregular processing, reduces in cardinality. Since *Sum_of_Odds* is a nonhelper function, its result is computed, paired with the respective range variable and *produced* in the universe:

U = { <Sum, [10]> }

The eventive construct also allows for the processing of *integrity constraints.* Rather than enabling nonhelper functions based upon the availability of domain variables in the universe, constraint functions are enabled based upon the availability of a universe *U* and its successor *U'*. Thus, transition constraints can be stated exactly one time and imposed automatically whenever there is a change to a universe.

In both the regular and irregular processing of nonscalars, one begins with a nonscalar and (typically) *reduces in the dimension* of the nonscalar (in the case of regular processing) or *reduces in cardinality* (in the case of irregular processing). There are times, when it is necessary to *expand* a nonscalar, increasing the nonscalar in dimension and/or cardinality. *SequenceL* possesses a ***generative*** construct which is used to expand nonscalars. Its basic form is to provide an upper and lower boundary for the term to be generated and an optional membership condition for items to be generated. For example, the term *[[1],...,[5]]* evaluates to *[[1],[2],[3],[4],[5]]* in *SequenceL* while *[[0],[1],...,=([!,+([pred(pred(!)),pred(!)])]),...,[20]]* evaluates to *[0],[1],[1],[2],[3],[5],[8],[13]].* It seems clear, that there are four ways to expand:

From left to right:
(lower bound) (membership condition) (upper bound)
From right to left:
(lower bound) (membership condition) (upper bound)
From outside-in: (e.g., convergence problems)

(lower bound) (membership condition) (upper bound)
From inside-out:
(lower bound) (membership condition) (upper bound)

The denotational semantics of *SequenceL* were completed in 1993. A prototype interpreter was completed in 1994. A proof that *SequenceL* is equivalent to the Universal Turing Machine was completed early in 1995.[F95] Many of these results are reported in earlier papers [C90-94, CG91]

5. Specifying Integrated Software

A second problem has to do with the integration of off-the-shelf tools and exploratory languages to form an appropriate software architecture such as is presented in Figure 1. Excellent off-the-shelf software exists to provide the image processing, networking, database, and geographical information system functionalities. The concern that we, and many others, face is our level of ignorance about the combined behaviors of resulting integrated systems. We are forced to (a). rely on the experiences of others who have combined some subset of these systems already or (b). put together software products and then stand back and observe what happens.

Facing this prospect has caused us to pause and wonder about the feasibility of identifying or developing a single specification language that could be used to describe the functional behavior of existing software products. A language such as this would need to have a precise semantic and be reasonably compact and easy to use. Based upon the language, it might be possible to develop an automated tool that could help one analyze the likely result of combining different software packages. Consider the following example.

Suppose one wished to combine a very simple relational database management system with a networking system. Assume that the database management system's interface is limited to a simple query language consisting of the relational algebra primitives: *union, difference, cartesian product, select,* and *join.* Assume that the network package consists of transaction primitives *post* and *receive.* Now imagine that both sets of transaction primitives have been defined precisely in some specification language. As an exercise, one could easily do this in Prolog. One can imagine the following combinations:

NETWORK	X	DATABASE
post		*difference*
post		*union*
post		*cartesian product*
receive		*select*
receive		*join*

Furthermore, one can imagine the combination of database transactions. For example, one might wish to select some set of tuples resulting from a cartesian product via the network (i.e., an interaction of *network x (database x database)*). In any case, if a simulator based upon the specification language were available, it might be possible to study the functionality that results from combining these two products. Furthermore, analysis tools could be developed to study the interactions as well. For example, one might wish to study the problems that arise with multiple users (e.g., *race* and *deadlock*), or those that occur due to network or database failures. The basic idea is that the specifications of the products could be studied at a fine level of detail to provide much greater predictability to the actual integration of tools.

The long term goal of a research effort to develop a specification language for the integration of tools would likely be to move the state of the practice towards a standard language. Suppose, in addition to receiving the normal set of brochures concerning a software product, one could *ftp* a specification of the product in this standard language. We intend to investigate the use of languages such as PSDL [BL90], MPL [W92], and RAPIDE [LK95] for specifying the functionality of tools identified in Figure 1.

Obviously the identification or development of such a language is a daunting task. The array of high level software tools currently available is quite large. Many of these tools have hundreds of commands. For example, in most image processing packages there exist many commands that can be used in an orthogonal way - there are commands to allow for data retrieval, processing, refining, etc.

Many areas of software engineering possess expertise to contribute to the endeavor introduced above. The areas include software slicing and merging, specification languages, reverse engineering, integration, etc. We believe that this area of software integration is where many computer scientists and software engineers will work in the future. We further believe that this represents a fundamental area of research, possessing a significant practical import.

5. Conclusions

It is our belief that the "reverse" prototyping of software packages is a promising approach to the the analysis of how packages will behave in the context of large software architectures. In addition to the *SequenceL* language development activity, we intend to use its software architecture as a laboratory for the study of methods to analyze software architectures. Our initial investigations will involve the use of the CAPS prototyping tool. In the initial invetigations we intend to prototype the various software packages we purchase. We will then combine the prototypes of the packages to see what functionality ensues. Since we are likely to purchase more than one GIS and more than one image processing package, we should be capable of determining the relative merits of the competing packages with respect to the total functionality of our software architecture.

BIBLIOGRAPHY.

[B93] J.P. Banatre and D. Le Matayer, "Programming by Multiset Transformation," *CACM,* Vol. 36 No. 1, (January, 1993), pp.98-111.

[BL90] V. Berzins and Luqi, "Languages for Specification, Design, and Prototyping", in P. Ng and R. Yeh (eds.), *Modern Software Engineering Foundations and Current Perspectives*, Van Nostrand Reinhold, pp. 83-118 (1990).

[C90] D.E. Cooke, "Towards a Formalism To Produce a Programmer Assistant CASE Tool," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2 No. 3, September, 1990, pp. 320-326.

[CG91] D.E. Cooke and A. Gates, "On the Development of a Method to Synthesize Programs from Requirement Specifications," *International Journal on Software Engineering and Knowledge Engineering*, Vol 1 No 1, (March, 1991) pp. 21-38.

[C92] Daniel E. Cooke, "An Issue of the Next Generation of Problem Solving Environments," *Journal of Systems Integration*, Vol 1(2), (February, 1992) pp. 39-52.

[C93] Daniel E. Cooke, "Possible Effects of the Next Generation Programming Language on the Software Process Model," *International Journal on Software Engineering and Knowledge Engineering*, Vol 3 No 3, (September, 1993) pp. 383-399.

[C94] Daniel E. Cooke, "Towards a Formalism for Program Generation," for the Air Force Office of Scientific Research, #F49620-93-1-0152, February, 1994.

[F95] B. Friesen, "The Universality of BagL," Masters Thesis, University of Texas at El

Paso, May, 1995.
[G85] D.Gelernter, "Generative Communications in Linda", *ACM Transactions on Programming Languages and Systems*, 7(1), pp. 80-112 (1985).
[HLS92] Chris Hankin, Daniel Le Metayer, and David Sands, "A Calculus of Gamma Programs," Publication Interne no 674, Juillet, 1992, IRISA, France.
[LK95] D.C. Luckham, J.J. Kenney, et. al. "Specification and Analysis of System Architecture Using Rapide," *IEEE Transactions on Software Engineering,* Vol. 21 No. 4, (April, 1995) pp. 336-355.
[M91] Milne, B.T. 1991. "Lessons from Applying Fractal Models to Landscape Patterns." In Quantitative Methods in Landscape Ecology, M.G. Turner and R.H. Gardner, eds. Springer-Verlag, New York, 199-235.
[O92] Ohkade, N. 1992. "A Real Time Algorithm for Fractal Analysis and Its Application to the Early Detection of Epileptic Seizures." MSEE Thesis. Department of Electrical and Computer Engineering, The University of Texas at El Paso.
[S86] Stanley, H.E. 1986. "Form: An Introduction to Self-Similarity and Fractal Behavior." In On Growth and Form: Fractal and Non-Fractal Patterns in Physics, eds. H.E. Stanley and N. Ostrowski, eds. Martinus Nijhoff, Boston, 21-53.
[S93] Starks,S.A., J.W. Hamilton and N. Okhade, "An automated approach to trend monitoring based upon fractal analysis," in *Intelligent Robots and Computer Vision XII: Algorithms and Techniques*, David P. Casasent, Editor, Proc. SPIE 2055, pp. 44-49 (1993).
[W92] G. Wiederhold, P. Wegner, S. Ceri "Toward Megaprogramming," *CACM,* Vol. 35 No. 11, (November, 1992) pp. 89-99.

---

1 A reducing construct, given an input D, will produce a result R where R is no larger (in dimension or cardinality) than D. E.G., Given a construct that reduces in cardinality: $D \rightarrow R$, the following relation holds after D is mapped to R: $|D| \geq |R|$

2 This operator represents a *SequenceL* evaluation step. The use of » » indicates many *SequenceL* steps.