A. INTRODUCTION

This paper describes patternForth, an extension of the Forth language to handle problems of string manipulation and pattern matching, in real time, for industrial control applications.

1. The Problem(s)

Some anticipated applications illustrate the need for this project, and the reasons why previous work is unsuitable.

a) Alarm analysis for nuclear reactors. Fault conditions in nuclear reactors, and many other process control situations, can cause a multitude of alarm signals. Also, any particular alarm signal can be caused by many different faults. The specific fault is identified only by a unique combination of alarms. While some work has been done [GAR89] on the use of expert systems for this analysis, it is essentially a problem of pattern recognition. This requires a pattern processing language integrated into an industrial control environment.

b) Integration of pattern matching into expert systems. Other expert systems could be more easily specified or more efficiently executed by having pattern specifications as expert system "rules." We could add pattern processing to an expert language, or add rule evaluation to a pattern matching language. Forth has been used as a base for expert systems [TRA85a,PAR83,SAN87,TOW86]; if we also add pattern matching to the Forth language, a pattern analyzing expert system follows naturally.

c) Text analysis in Forth programs. There is a substantial body of Forth code in existence today, serving a wide variety of purposes. Many of these -- particularly those involving human interface -- could benefit from the availability of a Forth-compatible pattern matching package.

2. Perceived Requirements

The example applications noted above define a set of requirements for this language:

a) Pattern and String Processing. All of the applications cited involve some form of pattern matching and string manipulation. This is the first "theme" of this project. Subsidiary requirements:

* Variable length strings. Pattern matching problems rarely involve text of predictable, fixed length.

* Dynamic memory allocation. Because of the high volume of string "traffic" -- i.e., the frequency with which strings are created and destroyed -- it is not feasible to pre-allocate string storage. It must be possible to request storage when it is needed, and release it for other uses when it is no longer needed.

b) Industrial Control. The second "theme" of this project is the use of pattern processing in industrial and process control environments. This is a radical departure from previous pattern matching work. Industrial control carries its own set of special requirements:

* Real time. Speed is of the essence in control environments; all of the functions of the computer system must execute either very quickly (most preferable), or in a predictable time (acceptable). Randomly occuring long delays in processing are anathema.

* Interactive. The computer must operate on a continuous stream of input and output data. "Batch" processing is useless in real-time control situations.

* I/O intensive. The data used by a process controller is not presented in disk files. Usually, the control program must interact directly with peripheral devices to acquire, format, and output the data. The language must support direct access to the I/O hardware, in contrast to most programming environments where the system resources must be protected from uncontrolled access.

* Nontext data. Very little industrial data is in the form of ASCII text. The pattern matching techniques for text must be extended to the realm of arbitrary binary data.

* Small computers. Mainframe computers are rarely used in industrial control these days, because of their cost and strict environmental requirements. The object programs must run on a microcomputer. (Given the current state of the art, this is not a severe restriction.)

c) Forth based. The use of the Forth language is often an explicit requirement. Forth is frequently chosen for process control problems, and it is finding increasing favor elsewhere as software development costs rise, because of the enormous increases it can offer in programmer productivity.

3. Previous languages

These problems have been addressed before. Some languages which incorporate elements of string processing or pattern matching are described here.

a) SNOBOL. Developed in the early 1960's, this is the examplar of pattern matching languages, and the language which we seek to emulate. SNOBOL's limitations are weak I/O and restriction to text data; no doubt due to its origins in a batch processing environment. Although it was not designed for real-time processing, much effort has gone into optimizing SNOBOL's pattern matcher.

b) Icon. This is the successor to SNOBOL. It uses a syntax similar to the C language. Pattern matching is less an explicit function than a capability. Patterns are described in a "procedural" fashion, and matched using Icon's goal-directed evaluation. Icon is much more powerful and flexible than SNOBOL, allowing (for example) the construction of list processors and expert system engines. Its "generator" concept is a powerful tool for the programming of goal-directed problems.

c) LISP. This is the original and foremost "list processing" language. It has sophisticated memory management and list manipulation capabilities. Strings are not a distinct data type, but rather are treated as lists. LISP shares with Forth the advantage of being an extensible language, and could be "tailored" to provide the desired functions. Unfortunately, LISP is not a real-time language; only with dedicated hardware can LISP perform quickly enough for industrial control. (LISP machines have been used for some work in alarm pattern analysis [GAR89].)

d) Forth. Forth is a language explicitly designed for real-time control programming. Efficiency is its hallmark, both in speed and in memory utilization. Forth provides a rich set of operators for manipulating the machine hardware. Its extensible nature allows the addition of new features and new language constructs. Forth includes practically no support for strings or memory management.

It is worthy of note that Forth has been extended to emulate LISP [TRA85a,TRA85b, TOW86], Prolog [TOW86], and Pascal [ZIM81].