PRODOC is a professional, self-documenting program generator and productivity tool which provides a revolutionary new programming environment for both professional programmers and neophytes alike. This intelligent system is based on the latest advances in understanding human cognition and automatically generates executable code from either high-level descriptions or Pascal Source code, all input in the form of self-documenting flow diagrams.

Neophytes will normally construct FLOW diagrams entirely from high-level descriptions, such as ADD (number1, number2, sum) (i.e., “add number 1 to number 2 and put the result in sum”), or CIRCLE (x-coordinate, y-coordinate, radius) (i.e., “draw a circle with center at point x,y and given radius”), which are directly meaningful to the user. In effect, PRODOC makes computers think the way people do rather than the reverse as is typically the case.

Equally important, PRODOC’s high level descriptions may be customized to meet the special requirements of particular application areas, such as engineering, medicine, general business, etc. Further enhancing the utility of PRODOC’s flow language is the fact that it is Pascal compatible. Hence, professional programmers may include traditional Pascal Source Code as desired and are not restricted to the high level flow language.

PRODOC is designed so that it can easily be extended. In future work, for example, it could be used as a basis for a more sophisticated product in which not only the computer code would be
generated automatically, but the high-level flow diagrams as well. This would be accomplished by having the user simply solve sample problems as he would like the computer to solve them. From this information, proprietary designs would allow the computer to infer the general procedural description (e.g., as a flow diagram) as well as generate the computer code underlying it.

PRODOC also provides a potential basis for a general purpose problem solver and learning system. In this case, the user would simply present a problem to the computer and PRODOC would figure out entirely on its own what kinds of procedures are needed and derive those procedures in order to solve the problem. In effect, proprietary designs would allow construction of a system able to learn — effectively to get progressively smarter as it is exposed to broader and broader varieties of problems. In short, the PRODOC line and its successors are not likely to be superceded in the near future.

More immediately, the current version of PRODOC, together with customized variations, could serve a variety of needs. In order to properly understand these needs, let us first list some of the things which the current version makes possible. PRODOC can be used to:

1. design, document and implement flow diagrams directly on the screen. (PRODOC uses a particular variant, called the "Scandura FLOWform," which is very easy to read.) The user may input ordinary English statements, high-level, user-friendly routines in an available library, program in any computer language (e.g., PASCAL), or use combinations thereof. To the best of our knowledge, most of these capabilities are unique.

2. interpret (execute) individual FLOWforms at any level of refinement, from the top level to the bottom. It is even possible to execute FLOWforms which include both English statements executable only by humans (e.g., "shut the door") and statements which can be directly interpreted (and executed) by the computer. Again, this is a unique feature.

3. automatically generate source code from FLOWforms which can be compiled for more efficient execution. The current version of PRODOC provides extra support for MICROSOFT Pascal and Borland International's TurboPascal. In this case all declarations, BEGIN...ENDS and other syntactic requirements are generated automatically so the user can concentrate on basic concepts without having to worry about sundry peripheral matters. Pascal also may be mixed arbitrarily with library routines. Future enhancements of PRODOC will provide full syntactic support for other languages such as BASIC, COHOL, Fortran and C. Other source code generators exist but none of them are as easy to use, are based on highly familiar FLOWforms, cater to NONprogrammers as well as programmers or are self-documenting.

4. program in any number of very high level (so called fourth generation) languages. These languages consist of libraries of rules (software routines), including various operations, conditions and input and output (e.g., display) routines. The library rules all have a common, easy-to-learn syntax which makes transparent to the user a multitude of distinctions, such as data types, text/graphics modes, etc. to which the programmer must otherwise painstakingly attend.

5. create any number of custom versions of PRODOC, using specially designed libraries, for special application areas (e.g., in engineering, medicine, business). We know of no other productivity tool which makes this feasible at all, much less so in such a general, automatic manner. Unlike the other features, option 5 will require limited access to PRODOC source code.

THEORETICAL BACKGROUND

So far we have discussed what PRODOC does and how it operates. By way of conclusion, I would like to say a few words about the theoretical foundations from which PRODOC was derived.

For two and a half decades, Artificial Intelligence (AI) researchers have struggled with the problem of how best to conceptualize, design and construct human-like intelligent systems — and some very promising results have been obtained. Throughout the 1960's, solutions were sought largely in logic and mathematics via systematic analyses of large state spaces. Eventually, these proved too inefficient for most complex problems and beginning in the 1970's, AI research came to be characterized by clever programming techniques and heuristic concepts — these were designed to overcome the combinatorial explosion problem. The basic idea was to capture critical essentials of human thought processes.

Most AI projects, however, have had their own idiosyncrasies.
Research in the field has not always been as cumulative as might be desired. Since AI systems have been implemented primarily in LISP or PROLOG, it is no surprise that the way most contemporary AI systems function has as much to do with the structure of these languages as with human thought.

Hence, the research which has characterized AI to date has been of uneven quality and often noncumulative. It seems unlikely that largely formalistic or intuitive techniques will ever be sufficient. With this in mind, Intelligent Micro Systems, Inc. (IMS) has adopted a third generation approach to the problems of AI. Specifically, IMS has been engaged in the construction of intelligent systems — based on more comprehensive and generalizable cognitive theory.

PRODOC, in particular, has been inspired by the rule construct in the Structural Learning Theory (SLT). As defined by Scandura (e.g., 1970, 1981), a rule recall is a triple, consisting of a domain, range and procedure. Procedures constitute step-by-step prescriptions for carrying out processes. In general, they are constructed from operations, conditions and input/output elements, the order and nature of which are arbitrary as long as each element can be unambiguously carried out. The procedures in SLT, however, are restricted in two important ways. First, SLT procedures are strictly structured in the sense that they are composed entirely of substructures of one of three forms: (a) a sequence of operations; (b) a selection among two or more operations, one of which may be vacuous; and (c) an iteration or loop, in which one operation is applied as long as some condition is satisfied.

Second, recursive operations are not allowed if the procedure is to be interpretable. (Recursive operations written in Pascal, however, may be compiled.) This restriction is consistent with the SLT where the role of recursion is reserved exclusively for a general purpose "goal-switching" control mechanism which determines which rules to use and when.

Rule domains and ranges are ordered sets whose elements in turn may themselves be ordered sets. Thus, for example, we might have a domain which consists of three elements, the first and third of which are also ordered sets. In turn, the second element of the first element might also be an ordered set. This parallels the hierarchical structure of procedures where successive refinement is possible. In effect, domains, ranges and procedures can all be represented at an arbitrary level of detail.

Because of its central role as the basic concept in a general theory of cognitive learning, the notion of a rule is extensible in a number of very important ways. In future work it will be used as a basis for systems in which not only the computer code would be generated automatically, but the high-level descriptions as well. This would be accomplished by having the user simply solve sample problems as he would like the computer to solve them. From this information, our designs would allow the computer to generate flow charts as well as generate the computer code.

The SLT also provides a potential basis for a general purpose problem solver and learning system. In this case, the user would simply present a problem to the computer and the system would figure out entirely on its own what kinds of procedures are needed and derive those procedures in order to solve the problem. In effect, we could construct systems able to learn broader and broader varieties of problems.

References

Scandura, J. M. Problem solving in schools and beyond: Transitions from the naive to the neophyte to the master. Educational Psychologist, 1981, 16, 139–150.