GRISWOLD81
Griswold, R.E., Hanson, D.R., and Korb, J.T.; Generators in Icon; ACM TOPLAS 3, 2 (April 1981) pp. 144-161.

Icon is an interesting new language for non-numerical computation based on SNOBOL4 and SL5. It has concise, expressive features, run-time flexibility, support for typed and untyped identifiers, heterogeneous structures, and automatic type conversions. The syntax is like Pascal but Icon supports backtracking and the novel "generators" that make it applicable to string processing, combinatorial search problems, and probably AI. The motivation for Icon was to provide the power of SNOBOL4 for pattern matching in a more general way so that the backtracking and other powerful operations can be used on more than just strings. Also, unlike SNOBOL4, Icon is intended to be production quality so there are some restrictions for efficiency.

The central concept in Icon is the generator. A generator is an expression that can return more than one value. The first value is returned, and if backtracking is needed, then the second value is tried, etc. The simplest generator is of the form " 1 | 2 | 3 " which returns 1 first then 2 if needed, etc. Another generator is " e1 to e2 by e3 " (where the by clause is optional). Finally, "!x" goes through all the elements of the aggregate (list, array, etc) "x". Generators are a powerful feature. For example, "mod(n, p := 2|3|5|7|11) = 0 " tests to see if n is divisible by any of the first 5 prime numbers. If so, p is assigned the divisor and the expression returns "true". If not, then the expression returns "fail". There are many other cases where generators can eliminate explicit loops. For example, "f(i:=1 to 5, j:=1 to 5)" keeps evaluating f with different arguments as long as it fails. The "every" prefix insures that all values of the generator are used, so "every i:= 1 to 10 do" replaces the conventional FOR loop. This is much more general, however, since the loop might be "every 1|3|5|16 do", or "every !x do", etc. Explicit loops may also be eliminated as in "every sum := sum + !x". There are a few built-in string generators, such as Find, UpTo, etc., which make pattern matching easy. There are also scoping mechanisms so that irrelevant generators can be easily eliminated to control backtracking.

Applications can define their own generators using the "suspend" command. This is an alternative form of "return" that allows a sequence of values to be returned. This is similar, but more general, than the iterator construct of CLU. When backtracking is performed, side-effects are usually not undone; only control is backtracked. There is a reversible assignment operator, however. This might be used, for example, to restore a default value if a read fails (returns a string of length 0): "size(param <- read()) ~= 0". Icon also has expandable arrays, tables, and strings.

Icon runs on a number of machines and has proved to be useful for string processing and other applications. Its main contribution is the unified linguistic facilities where goal-directed evaluation is an integral part. Backtracking is provided, but its scope is easily controlled by the programmer for maximum flexibility and efficiency. Icon seems like a very interesting language and it seems like it might be applicable to many different domains. This paper is well written and succeeds in demonstrating the novel aspects of Icon.

Another article about Icon is Ralph E. Griswold, David R. Hanson, and John T. Korb, "The Icon Programming Language: An Overview," ACM SIGPLAN Notices. 14, 4 (April 1979) pp. 18-31, which has more background for the language and details on the syntax and general features. The generators are the most interesting part, however, and there is sufficient background in the TOPLAS article for them to be understood.