Automatic programming

The term Automatic Programming is ill-defined. It means many different things to many people. It has covered everything from Fortran Compilers in the mid-50's to futuristic visions of machines which completely program themselves to accomplish a given goal, or are even capable of generating those goals themselves.

The three Automatic Programming sessions in this conference clearly cannot cover such a broad spectrum. Instead, they are focused on a set of small research projects each attempting to expand the computer's involvement in the programming process. Surely, such continued expansion of computer involvement in the programming process is the central thrust of all the efforts in this field.

The current state of this field can best be accessed by noting that programming is still largely a manual process. Significant progress has been made over the years, but it has largely been concentrated in the final stages of the programming process (such as language translators, test case generators, and debugging tools). What facilities exist which help us formulate and design systems, study tradeoffs between alternative designs, foresee the implications of design and implementation decisions, understand the effects of modifications, predict system performance, foresee system bottlenecks, or even prevent the introduction of bugs during implementation?

Such tools basically do not exist. Instead, programmers and systems analysts deal with these issues in their own internal ways. These internal methods are not accessible for analysis by others, are error-prone, incomplete, and difficult to transfer from one individual to another.

This deficiency has been recognized and responded to by the development of a set (actually several competing sets) of techniques for MANAGING the programming process. These techniques attempt to control the complexity of programming by advocating certain practices and styles while proscribing others, and by developing languages which encourage and/or enforce such direction. Certainly such endeavors have significantly improved our ability to control and manage these design and formulation processes occurring within our heads.

Such manual techniques can only go so far. The regularized (structured) environment that they have created must be augmented by various tools which
record the stepwise development of these processes as they occur within our heads, expose the decisions we are making, and enable us to see the implications of those decisions.

The challenge for the field of Automatic Programming is to extend the programming environment by constructing such tools which expose this stepwise development process and enable each stage to be analyzed, as the basis for the next step, without impairing the formulators' design flexibility. To accomplish this, there must exist a clear separation between a decision of what step to take next, which the formulator must control, and the carrying out of that decision, which the tool must perform to ensure that it is being accurately accomplished. In this way the integrity of the system being formulated is maintained as it moves from stage to stage. These tools thus ensure system integrity between formulation stages by automating the "clerical" operations which transform one stage into the next after a decision has been reached. Since a very large number of decisions must be made in any formulation process and some of them are very detailed, there may well exist some portion which the formulator relegates to the Automatic Programming tool. These Automatic Programming tools must also provide some analysis capabilities which enable the formulator to determine which decision to make next and how to make that decision.

All of the prototype tools covered in these three Automatic Programming sessions match this paradigm. They differ in the portion of the formulation process they address, the analysis capabilities available at each stage, and the degree of control the formulator maintains over the decisions which get made.

The first session concentrates on tools which address the process of specifying a program, while the second is composed of those which address the implementation of such a specified system. These two sessions are not intended to present new research results (these have been reported in various specialized conferences and workshops), but rather, to bring together and highlight representative work in this field for the NCC audience.

The final session, a panel, will attempt to assess the extent and timeframe of the impact of such work within the computing community.