XIII

# SNOBOL Session

*Chairman: JAN Lee*
*Speaker: Ralph E. Griswold*

## PAPER: A HISTORY OF THE SNOBOL PROGRAMMING LANGUAGES

*Ralph E. Griswold*

Department of Computer Science
The University of Arizona

Development of the SNOBOL language began in 1962. It was followed by SNOBOL2, SNOBOL3, and SNOBOL4. Except for SNOBOL2 and SNOBOL3 (which were closely related), the others differ substantially and hence are more properly considered separate languages than versions of one language. In this paper historical emphasis is placed on the original language, SNOBOL, although important aspects of the subsequent languages are covered.

### 1. Background

#### 1.1. Setting

The initiative for the development of a new string manipulation language that was to become SNOBOL arose in the Programming Research Studies Department at Bell Telephone Laboratories, Incorporated (BTL). This organization was originally situated at Whippany, New Jersey, but moved to the new Holmdel, New Jersey, Laboratory in mid-1962. The department was part of the Switching Engineering Laboratory, which was mainly responsible at that time for advanced planning and engineering for the Electronic Switching System (ESS) telephone offices. The department itself, although not part of the Research Division of BTL, was concerned with computer-related research and its members were relatively isolated from ESS concerns. Departmental work was in areas of automata theory (Lee, 1960, 1961b), graph analysis (Lee, 1961a; Griswold and Polonsky, 1962;



ISBN 0-12-745040-8

Griswold, 1962), associative processors (Lee and Paull, 1963), and high-level programming languages for formula manipulation (Lee *et al.*, 1962).

To understand what follows, the environment within the department needs to be understood. The department was small by BTL standards (six persons) and had no internal supervisory structure. The department head, C. Y. (Chester) Lee was mainly concerned with his own research and other members of the department worked relatively independently, either individually or in small groups that formed naturally. In 1962 Ivan P. Polonsky and I were working with the SCL programming language that had been developed at BTL by Chester (Lee *et al.*, 1962) and were dividing our efforts between problem solving and the application of SCL to such problems as symbolic integration, factoring of multivariate polynomials, and the analysis of Markov chains. David J. Farber was also in the department, but he was more concerned with operating systems and software itself. At that time, he was also secretary of SHARE and was involved in many activities inside and outside BTL that were not directly related to the other research going on in that department.

Although SCL had a number of interesting features, it had notable deficiencies: poor performance, a severely limited data region, and an intricate, low-level syntax that required lengthy programs for realizing simple operations. As Ivan and I became increasingly frustrated with the limitations of SCL, Dave obtained COMIT (Yngve, 1958; MIT Press, 1962) for our use. Although COMIT was interesting, its difficult syntax and orientation toward natural-language processing made it unsuitable for formula manipulation and graph analysis. In the fall of 1962, the decision to attempt the design of a new programming language came spontaneously from Dave, Ivan, and myself. We simply felt that we could produce a language that was superior, for our purposes, to any one that existed at that time. Because of the relative independence of the members of the department and the absence of any formal channels of communication or approval, the early work that led to SNOBOL went on without the specific knowledge of Chester and, in fact, in an air of considerable secrecy. At the same time, Chester was working on an improved version of SCL, although we were not then aware of his work either.

These two independent movements to develop a new programming language surfaced with Chester's presentation of his new version of SCL in November 1962. While his proposal added a number of new facilities to SCL, it retained and expanded the features that had led to our divergent work. When we subsequently revealed our design (which was considerably more developed than Chester's) the conflict was evident: the department could not support the development of two languages. Without our help, Chester did not have the resources to implement his proposal. Since Chester was head of the department, we could not continue our work if Chester directed us not to. While Chester did not stop our work, there was considerable tension within the department in the ensuing months.

### 1.2. Initial Design

The group that was developing SNOBOL worked out its own internal relationships on the basis of the personalities and the qualifications of the persons involved. Dave was the acknowledged leader. He was the only member of the group with substantial software development experience and with contacts in the professional computing community. I had virtually no computing experience prior to coming to BTL and Ivan's background was largely in application areas rather than in software development.

The group operated autonomously and without outside direction. Dave provided an interface with other computing organizations within BTL, most notably the computer center and the research department at the Murray Hill Laboratory. His discussions with persons there—M. Douglas McIlroy, Robert Morris, and Victor A. Vyssotsky—undoubtedly had an indirect influence on the design and implementation of SNOBOL.

It should be noted that Dave viewed himself primarily as a language designer and implementor, while Ivan and I initially viewed ourselves primarily as eventual users. This view changed over a period of time as SNOBOL became more important to us than the problems it was intitially designed to solve. All members of the group regarded the effort as tentative, with its ultimate direction depending on the success of the initial efforts.

### 1.3. The Implementation

Consideration of how the implementation might be handled was present in the early language design stage. When the initial design was satisfactory to the group, actual implementation became the next goal. A trial implementation was undertaken on an IBM 7090 early in 1963. At that time the group felt that a running system was essential in order to determine whether or not the new language was worthwhile. We did not think that a large initial investment was justified, reasoning that if the language proved valuable, more design and another implementation would be required.

Dave felt that a true compiler for a language like SNOBOL was impossible and that an interpretive system would give us the flexibility to implement the kinds of language features we wanted. Source-language statements were translated into intermediate forms of flags and pointers suitable for efficient interpretation. This style of interpretation provided the model for future implementations. With its flexibility, it also provided a vehicle suitable for developing esoteric language features.

Dave sketched out the organization of an interpretive system and wrote the compiler and interpreter. The attitude toward the implementation is exemplified by the compiler, which Dave literally coded in macro FAP on the back of a business envelope one evening. (It has been said that the compiler *ran* as if it had been coded on the back of an envelope.) Doug McIlroy's string manipulation macros (McIlroy, 1962) were used extensively in the initial implementation and were to figure prominently in subsequent work. Ivan and I implemented the pattern-matching portion of the interpreter. This proved to be a challenge, since I had no prior assembly-language programming experience and Ivan's experience was limited. Laura White Noll assisted the group by implementing the symbol table and storage management routines, again relying on work done by Vic Vyssotsky and Doug McIlroy on scatter storage techniques (McIlroy, 1962, 1963; Morris, 1968).

The short period of time required to achieve the first usable implementation of SNOBOL—about three weeks—was a matter of pride within the group and was held as proof that the implementation of a programming language need not be time consuming and costly.

While the original implementation was not of the "toy" variety, it nonetheless required considerable incremental improvement and debugging, especially in the area of storage management. By the summer of 1963, however, the implementation was in extensive use within BTL, mainly in the Research Department at Murray Hill and the Electronic Switching Systems groups at Holmdel.

### 1.4. Reaction to SNOBOL

Dave, Ivan, and I were the first users of SNOBOL, although use spread rapidly through BTL. The earliest recorded applications were in the evaluation of algebraic expressions (Griswold, 1963c), a program to convert network descriptions into FORTRAN programs that performed statistical analyses of these networks (Faulhaber, 1963), a program to convert descriptions of ESS data into IPL-V programs that generated the corresponding assembly-language code to provide data regions for ESS programs (Griswold and Polonsky, 1963b), and a FORTRAN compiler (Farber, 1963a).

The initial reaction of users to SNOBOL was surprisingly enthusiastic. Programmers at BTL quickly learned the new language and began using it for a variety of problems. By late 1963 and early 1964, SNOBOL had been used within BTL for text generation (Manacher, 1963), graph analysis (Magnani, 1964), syntax analysis (Griswold, 1964a), and simulation of automata (Faulhaber, 1964). It was also being considered for application to several engineering problems of interest to the Bell System. There was no question that SNOBOL was a success. Chester acknowledged its value (Lee, 1963) and permitted work on SNOBOL to continue.

The first official announcement of SNOBOL outside BTL came in a talk by Dave at the University of Michigan in October 1963 (Farber, 1963b). Other talks followed and information spread by word of mouth. With distribution of the SNOBOL system, interest increased rapidly. Obviously a long pent-up need for such a programming language had been satisfied. Programmers began using SNOBOL for problems which they were formerly unwilling to undertake. Examples were programs to generate other programs (Faulhaber, 1963; Griswold and Polonsky, 1963b) and "quick-and-dirty" experimental compilers (Farber, 1963a).

### 1.5. The Problem of a Name

While SNOBOL is the only name for the language that most persons have ever heard, several other names were used tentatively and the final choice came with some difficulty. The language was initially called SCL7, reflecting its origins. At that time SCL was officially at version 5 (SCL5) and the name SCL7 reflected a feeling of advancement beyond the next potential version. Actually the new language was very much different from SCL. With Chester's separate language design and an increasing disaffection with SCL on our part, we sought a new name.

The first choice was the acronym SEXI (*S*tring *EX*pression *I*nterpreter). The first implementation of SNOBOL in fact printed this name on output listings (Griswold, 1963c). The name SEXI produced some problems, however. Dave recalls submitting a program deck with the job card comment SEXI FARBER as per BTL standards, to which the I/O clerk is said to have responded, "that's what *you* think." In the original draft report, the typed name SCL7 was changed by hand to SEXI except in the opening sentence (Griswold, 1963b). In Chester's critique of this draft (Lee, 1963), he commented:

> I feel the use of a name such as SEXI may be justified if the language is so poor that it needs something to spice it up. Here we have an extremely good language. The use of SEXI may be interpreted as a lack of confidence on our part.

The draft report itself acknowledged the problem of a name in its last paragraph (Griswold, 1963a):

> A suitable name for such a string manipulation language is also badly needed . . . in addition to the two names referred to in this report [SCL7 and SEXI] . . . PENELOPE has been suggested, after the wife of Ulysses.

In fact the name PENELOPE was considered because of its graceful sound—the reference to Ulysses was an afterthought.

Despite a continued attachment to the name SEXI, we recognized that such a name was unlikely to gain approval from the authorities who controlled the release of information from BTL. Consequently the search for a new name began. This search was long and frustrating. We have since jokingly commented that it took longer to find a name than it did to design and implement the language (and that it was harder). The result was less satisfactory, as well, in most views.

We concentrated mainly on acronyms, working with words that described significant aspects of the language, such as "expression," "language," "manipulation," and "symbol." Hundreds of suggestions were made and rejected. I take personal credit and blame for the name SNOBOL. However, the name was quickly accepted by Dave and Ivan. As I recall, I came up with the name first and then put together the phrase from which it was supposedly derived—*S*tri*N*g *O*riented sym*BO*lic *L*anguage. This "pseudo-acronym" was intended to be a joke of sorts and a lampoon on the then current practice of selecting "cute" names for programming languages. Our humor was not appreciated and explanations of the origin of the name met with puzzled or pained expressions. With Steve Unger's comment, "that's *disgusting*," we suppressed the basis for the name SNOBOL until some years later when a publisher insisted on an explanation (Griswold, 1972a, p. ix).

The relatively lame humor exhibited in the choice of SNOBOL was not limited to the name of the language itself. Dave's error messages such as "ALL OUT OF SPACE, YELL FOR HELP" and "NO END STATEMENT, WHISPER FOR HELP" and the cryptic "RECOMPILE NOT ENABLED" quickly became wearisome to users.

Had we realized the extent to which SNOBOL would become popular, we probably would have selected a more dignified name. I recall that it was years before I could give a talk on SNOBOL without a sense of embarrassment about the name. SNOBOL has, of course, attracted many puns and bad jokes (e.g., the chance of a SNOBOL in hell). For some curious reason the lame humor of the name has been carried on by others. An example is Jim Gimpel's ICEBOL program for compressing SNOBOL programs (Griswold *et al.*, 1968b, pp. 197–202). Probably the worst pun is SPITBOL, *SP*eedy *I*mplemen*T*ation of SNO*BOL* (Dewar, 1971). Other examples are FASBOL (Santos, 1971), SNOFLAKE (Haight, 1970), ELFBOL (Gimpel and Hanson, 1973), SNOBAT (Silverston, 1976a), and most recently SLOBOL (Dalgleish, 1977).

A curious aspect of the choice of a name arose in the conflict with Chester between his language and ours and how the personnel would be affiliated. Implicitly acknowledging the success of the language, ultimately to be named SNOBOL, Chester offered to join our effort if we retained the name SCL but not if we picked another name. We decided to pick another name.

### 1.6. SNOBOL2 and SNOBOL3

A successor to SNOBOL was almost inevitable. Since there was no question of the demand for SNOBOL, issues in the original design and implementation had to be faced. The most notable design problem was the lack of any built-in function mechanism. As a result,

even the simplest computations often required contorted, laborious, and inefficient programming techniques. In addition, the original implementation, while workable, was not sufficiently complete and not robust enough for wide use.

SNOBOL2 included a few minor changes to SNOBOL and a number of built-in functions for string and numeric computation. In addition, a facility for adding separately compiled assembly-language functions was included (Farber *et al.*, 1965b). The implementation was mostly redone from the ground up but again for the IBM 7090. This time more extensive use was made of the string manipulation macros (McIlroy, 1962) and their repertoire was extended (Farber *et al.*, 1965a).

SNOBOL2 was placed in use at BTL in April of 1964 (Farber *et al.*, 1964c). Although SNOBOL2 was used briefly at Project MAC under CTSS (Corbató *et al.*, 1963), it was never generally distributed outside BTL.

From the inception of SNOBOL2, it was clear to us that it had a serious deficiency: the lack of a mechanism for programmer-defined functions. This mechanism was designed, but not implemented because of the lack of personnel resources. With the summer assignment of Lee C. Varian (then a student at Princeton University) we had the necessary support, and the implementation of SNOBOL3 was undertaken. Glenn K. Manacher, on loan from the research division at Murray Hill, assisted in the design and implementation. SNOBOL3 was running in July of 1964 (Griswold, 1964b), and it officially replaced SNOBOL2 at BTL in October (Griswold, 1964c).

### 1.7. A Refractory Period

SNOBOL3 was greeted enthusiastically in the user community and its distribution to IBM 7090/94 installations was widespread. As a result, we became more and more involved in distribution, maintenance, and documentation.

As a result of increased interaction with the computing community after the release of SNOBOL3, we received many suggestions for changes and additions to the language. One of the evident deficiencies of SNOBOL3 was its lack of facilities for manipulating structures. Ivan and I began a series of investigations and experiments in this area using the external function mechanism of SNOBOL3 to add structure manipulation functions written in assembly language (Griswold and Polonsky, 1965; Griswold, 1965). While the results were interesting, they were inconclusive and did not provide clear direction for any substantial changes to SNOBOL3. Some related work continued, but language design and development were largely inactive. Unlike the progression from SNOBOL to SNOBOL2 and SNOBOL3, the progression to SNOBOL4 was not inevitable.

### 1.8. The Setting for SNOBOL4

This refractory period coincided with major changes in computing at BTL. In 1965 badly overloaded computing facilities and dissatisfaction with batch-mode operation, coupled with the imminence of the third-generation large-scale scientific computers, led to a corporate decision to revamp the computing facility and its mode of operation. The MULTICS project (Corbató and Vyssotsky, 1965) came into being, with large-scale interactive computing on GE 645s proposed as a replacement for the IBM 7094 batch operations then in use.

This major transition to new hardware of course implied major software development,

including development of a replacement for SNOBOL3. The reimplementation of SNOBOL3 was not an interesting or challenging project, but new ideas on language design coincided with this concern and led to the proposal for the development of SNOBOL4 on the new computer. We proposed to implement a SNOBOL-type language for the new computer, provided we could develop a new language as well. Our proposal was accepted, although no description of the new language was offered or requested.

In September 1965 a working group for production of GE 645 software was established within the local organization and James F. Poage was given the "technical responsibility for the implementation of SNOBOL IV [sic]" (Poage, 1965).

There was some reconsideration of an alternative for the name SNOBOL4. There were two main issues. On one hand, we expected SNOBOL4 to be substantially different from SNOBOL, SNOBOL2, and SNOBOL3. Hence there was an advantage to the selection of a completely new name to avoid the implication that SNOBOL4 was simply another version of SNOBOL3 with some minor changes. On the other hand, we felt the identification associated with the name SNOBOL would be valuable. We eventually made the decision to retain that identification.

### 1.9. Early SNOBOL4 Work

An early decision had to be made on the method to be used to implement SNOBOL4, since there was no GE 645 available for use. Most MULTICS-related GE 645 software development was done at Project MAC, originally using a 645 simulator running on a GE 635. EPL (Cambridge Information Systems Laboratory, 1968), an early version of PL/I, was used for most systems programming. Since our work on SNOBOL4 had low priority within the MULTICS project, and at Holmdel we were remote from the sites of major development work, we decided to do our original implementation of SNOBOL4 on our local IBM 7094, which had good, running software and to which we had access. We anticipated, of course, the need to transport our implementation to the new computer.

In retrospect, it is interesting to note that although SNOBOL4 was officially part of the MULTICS project, the affiliation was largely ignored by both sides. No schedules were established, no goals were projected, and no reports of documentation were ever requested or supplied.

Work on SNOBOL4 began in earnest in February 1966. The principals in the design and implementation of SNOBOL4 were Jim Poage, Ivan, and myself. Robert A. Yates served in a supporting role in the implementation of storage management routines and Al R. Breithaupt, on a temporary internship assignment, provided general programming support. By this time Dave Farber had other interests and responsibilities. He served primarily as an advisor on implementation and helped in providing support software. Although Jim Poage was the supervisor of the group developing SNOBOL4, I assumed de facto leadership of the technical aspects of the project.

Several important concerns affected the design of SNOBOL4. One was the need for a portable implementation that could be transferred to the GE 645 easily when the new computer became available. Another, more fundamental concern was our desire for more generality in the implementation. We expected SNOBOL4 to serve as a vehicle for language experimentation and to be sufficiently flexible that design and implementation could be concomitant—with new features tested experimentally in the context of a working system. From an implementation viewpoint, this led to the evolution of the machine-inde-

pendent macro language SIL (Griswold, 1970, 1972a) that had origins in the use of McIlroy's string macros in the early implementation of SNOBOL.

Efficiency had been a minor concern in SNOBOL, SNOBOL2, and SNOBOL3, both to the designers and to the users, until a number of large applications raised questions about costs and memory limitations. During the initial design period of SNOBOL4, the prospects offered by third-generation hardware were the subject of much discussion. With the promise of cheaper, faster processors and virtual memory, software planners were envisioning an environment in which efficiency considerations could be ignored and memory could be treated as virtually unlimited. While we were skeptical, we nonetheless decided to design SNOBOL4 as if speed and space were not serious concerns. We recognized from the beginning that the resulting system would be inefficient and we anticipated the need for more efficient implementations if SNOBOL4 became popular.

The philosophy of generality, flexibility, and concomitant design and implementation had a greater effect on SNOBOL4 than on earlier languages. While SNOBOL, SNOBOL2, and SNOBOL3 were largely designed prior to implementation and completed as identifiable projects, SNOBOL4 development extended over a period of years and passed through many stages as new features were added and old ones discarded.

The implementation of SNOBOL4 was a metamorphic process. The first versions were obtained by gradual conversion of SNOBOL3, using external functions for features (such as pattern matching) that were substantially different from those of SNOBOL3. By April 1966 (Griswold, 1966b) a running version was used experimentally. The emphasis on flexibility and generality is evidenced by the fact that this version permitted source-language redefinition of internal executive procedures. Even the meaning of literals could be changed during program execution.

By August 1966 (Griswold, 1966d), a relatively complete preliminary version of SNOBOL4 was working on the IBM 7094 and language design and implementation were proceeding concomitantly as planned. In the meantime, the BTL decision to convert to GE 645s under MULTICS had been reconsidered and some of the laboratories had decided to change to other computers. Most notable was the Indian Hill Laboratory's change to an IBM 360/67 under TSS. In June 1966 (Martellotto, 1966) the Indian Hill Laboratory expressed interest in a "SNOBOL compiler" for the 360/67. The decision of our own Holmdel Laboratory in 1966 to go to a 360 under OS shifted our own implementation concerns.

Our first use of a 360 was at Princeton University, where Lee Varian arranged for us to obtain access to their computer facilities and supported our first attempt at transporting SNOBOL4 to a new computer. This work began in October 1966 and a 360 version of SNOBOL4 was running by the end of the year. In the spring of 1967, a 360/40 was installed at Holmdel. Although the first SNOBOL4 system sent outside BTL in June 1967 was for the IBM 7094 (Griswold, 1967), development work had been shifted entirely to the IBM 360 by this time.

### 1.10. Continuing SNOBOL4 Development

With the first release of SNOBOL4 outside BTL, work continued on new features. Beginning in mid 1967 considerable support was given as well to transported implementations in progress by other persons for the CDC 6000, GE 635, UNIVAC 1108, RCA Spectra 70, and CDC 3600.

Employees from other BTL organizations who were on temporary assignment to gain

additional work experience augmented the personnel regularly assigned to the project. Paul D. Jensen and Bernard N. Dickman, who joined the project for the summer of 1967, designed and implemented tracing facilities (Dickman and Jensen, 1968). In the spring of 1968, Howard J. Strauss, also on temporary assignment, designed and implemented external functions (Strauss, 1968). James F. Gimpel was a contributor to continuing SNOBOL4 development, assisting in documentation, theoretical investigations of pattern matching (Gimpel, 1973a), and developing language extensions of his own (Gimpel, 1972). Michael D. Shapiro participated in the project intermittently from 1967, contributing ideas, testing the system, assisting with the documentation, and working on the CDC 6000 implementation (Shapiro, 1969).

The distribution of SNOBOL4 was substantially more widespread than that for SNOBOL3. By the end of 1967 (Griswold, 1968), some 58 IBM 360 systems had been distributed and documentation was being widely disseminated (Griswold *et al.*, 1967b). The decision to undertake extensive distribution was made by the implementors and I did most of the distribution myself, including copying magnetic tapes.

Despite the increased burden of distribution, correspondence, and maintenance, design and implementation continued. The first officially complete version was distributed in March 1968. Version 2, with substantial language changes, was distributed in December 1968 (Griswold *et al.*, 1968a), and Version 3, the last version released from BTL, was completed in November 1969 (Griswold *et al.*, 1971).

Subsequent work at BTL consisted of corrections and a substantial amount of documentation that was related to implementation and installation. In August 1971 I left BTL to join the faculty of the University of Arizona. At that time, active development of SNOBOL4 per se ceased, although Jim Gimpel at BTL continued his work on pattern matching (Gimpel, 1975), implementation techniques (Gimpel, 1974a,b), and the development of an efficient SNOBOL interpreter for the DEC-10 (Gimpel, 1973b). At the University of Arizona, work turned toward more research-oriented topics (Griswold, 1975c; Hallyburton, 1974; Druseikis and Griswold, 1973). Although some SNOBOL4 activity continues at various locations, it is now mostly concerned with implementations for new machines and with techniques for achieving greater efficiency.

### 1.11. Documentation

Until the first implementation of SNOBOL was working, little thought was given to documentation. The first attempt at documentation came in the form of a draft in March 1963 (Griswold, 1963a). The first complete document was an internal memorandum in May of that year (Farber *et al.*, 1963a), which was released in October for distribution outside BTL. A journal article based on this paper was submitted to the *Journal of the ACM* in October of 1963 and was published in January of 1964 (Farber *et al.*, 1964a). Rapid publication hastened the public awareness of SNOBOL. The manuscript had been distributed for review prior to submission and it was accepted the day after it was formally submitted (Farber *et al.*, 1963b; Hamming, 1963). The article appeared in print in less than three months.

The only documentation of SNOBOL2 was an internal draft memorandum (Farber *et al.*, 1964b). The lack of more formal documentation reflects the limited time that SNOBOL2 was in use.

Documentation for SNOBOL3 followed the pattern for SNOBOL—internal memo-

randa that were subsequently approved for release outside BTL as technical reports. A basic report describing the language (Farber *et al.*, 1964d) was followed by reports on the external function mechanism (Farber *et al.*, 1965b) and extensions to SNOBOL3 effected by use of the external function mechanism (Manacher, 1964; Manacher and Varian, 1964; Griswold and Varian, 1964; Griswold, 1965; Griswold and Polonsky, 1965). Official publication of a description of SNOBOL3 came in the form of a long article in the *Bell Systems Technical Journal* (Farber *et al.*, 1966). This journal was selected for the publication primarily because of its willingness to publish a paper of sufficient length to include a complete description of SNOBOL3. Prentice-Hall expressed interest in publishing a book on SNOBOL3, but after an investigation of the potential market, I discouraged this project. However, a SNOBOL3 primer was published by the MIT Press (Forte, 1967).

With SNOBOL4, documentation was taken much more seriously. Informal working papers on design and implementation were encouraged, but distribution was limited to those involved in the project. Several series of documents relating to the SNOBOL4 project were initiated for distribution outside BTL. These included installation notes and a series of technical reports (S4Ds) documenting the language and its implementation. The first S4Ds were concerned primarily with language descriptions (Griswold *et al.*, 1967b,c, 1968a). To date the S4D series has run to 56 documents, many of which have been revised repeatedly to reflect changes in SNOBOL4 (Griswold, 1978b).

By the time SNOBOL4 was available for general distribution, the language manual was finished (Griswold *et al.*, 1968a) and was being distributed by BTL. While distribution of earlier documentation had not been a significant problem, the demand for this manual was high. Before the publication of a book on SNOBOL4, over 2000 copies of the manual were distributed free of charge by BTL, and approval had been given for reprinting of over 1000 copies by organizations outside BTL.

In 1968 an adaptation of this manual was printed by Prentice-Hall (Griswold *et al.*, 1968b) and a revised, second edition (the "Green Book") was published in 1971 (Griswold *et al.*, 1971). This latter book remains the standard reference to the language. A primer more suitable for inexperienced programmers was published in 1973 (Griswold and Griswold, 1973), to be followed by other introductory books (Newsted, 1975; Maurer, 1976). Recently, more advanced books on the use of SNOBOL4 have appeared (Griswold, 1975a; Gimpel, 1976; Tharp, 1977). A book describing the implementation and especially its interaction with design appeared in 1972 (Griswold, 1972a).

Although there has never been an organized SNOBOL user's group, topical information has been published in the SNOBOL Bulletin of *SIGPLAN Notices* (Waite 1967–73) and the *SNOBOL4 Information Bulletin* which is issued aperiodically (Griswold, 1968–).

### 1.12. Support, Costs, and Schedules

Someone always wants to know how many man-years it took to develop a language and what the total cost was. Except in particularly highly structured and closely directed projects, such figures are difficult to determine at best and may be grossly misleading at worst. In addition, simple manpower estimates do not take into account the varying range of abilities of the persons involved.

For the SNOBOL languages, assignments of persons to the project were often informal or unofficial. Work was done spontaneously, motivation was generated within the group,

and schedules were either nonexistent or generated internally to ensure adequate progress.

Since the development of SNOBOL spanned such a short period of time, fairly accurate manpower estimates are possible for it. Dave, Ivan, and I were involved for about nine months from the inception of the design to the completion of the first implementation. In addition, Laura White Noll provided about four months of programming support. Allowing for our other responsibilities, approximately one and one-half man-years were involved altogether.

SNOBOL2 was also produced in a relatively short time with a manpower expenditure of about two man-years. SNOBOL2 blended into SNOBOL3, but perhaps an additional two man-years were expended before the first public version appeared, although the time spent on subsequent maintenance was significant.

For SNOBOL4, accurate figures are impossible to determine. Part of the problem lies in the continuous evolution of design and development of SNOBOL4 over a period of years, making it difficult to know when one aspect ended and another began. Another confusing aspect was the responsibilities of the personnel involved. These duties became increasingly complex as the project continued. Jim Poage and I had major management responsibilities unrelated to SNOBOL4. As a rough estimate, perhaps six man-years were expended before the first running version of SNOBOL4 was available at BTL.

BTL always provided good physical support for SNOBOL. As far as costs are concerned, the major item was computer time. Although there were internal computer budgets, the accounting was never carried to the level of the persons involved in SNOBOL. Costs of SNOBOL development probably were never identified as such but instead were lumped with other projects. Certainly we used the computer resources at our disposal lavishly. In the absence of budget constraints, we felt that computer time was (in some sense) cheaper than our time, and we did not hesitate to make repeated batch runs to correct small errors if that speeded our work. While it was a luxury to have virtually unlimited computer access, budgets were only one constraint. We frequently were hampered by batch-mode operation and poor turnaround time, especially in the period between 1966 to 1968 when the second-generation computer facilities had become saturated and the transition to third-generation facilities introduced inevitable problems. Many entries in the SNOBOL4 log indicate continued frustration with computer facilities to the point that information about design and implementation is meager by comparison (Griswold, 1966–1969).

With the development of SNOBOL4 into a substantially larger project than any of the earlier languages had been, support of documentation, distribution, and maintenance by a small group became an increasing problem. The use of computer systems to support the work became essential. Emphasis was placed on document preparation systems, and we became leaders in introducing this technology to BTL. We first used TEXT90 and then TEXT360 (LeSeur, 1969) when the Holmdel Laboratory converted to third-generation hardware. The installation of ATS (IBM, 1969) provided the first interactive system for our use, and substantially reduced the effort of document preparation. We also developed a considerable amount of our own software (implemented in SNOBOL4, of course), including document formatting languages with interfaces to line printers and photocomposition equipment (Noll, 1971). As a result, the first Prentice-Hall book was delivered to the publisher as line-printer output in camera-ready form and the second edition was photo-

composed. The use of camera-ready material significantly shortened the production period, and the use of a number of programs for tabulation and indexing produced unusually error-free copy for this kind of material.

Personnel support was another matter. While we generally favored a small, closely knit design and implementation group, there were many times when programming support and clerical help would have been welcome. What little help that was provided came in the form of temporary assignments: summer employees (as in the case of Lee Varian) or internal, rotational internships designed to broaden the background of employees by temporary assignments (Al Breithaupt and Howard Strauss were in this category).

SNOBOL4 produced our first serious manpower problem. Despite the enthusiasm with which SNOBOL4 was received and the accompanying publicity, no additional personnel support was available. Late in 1966, Jim Poage asked for additional personnel in programming support and clerical roles (Poage, 1966b). He listed 10 major areas in which work was needed and commented:

> It is unreasonable to expect that the people now working on SNOBOL4 (R. E. Griswold, I. P. Polonsky, and J. F. Poage) will be able to do all of the tasks listed, much less be willing to do them. Support for the SNOBOL4 effort is absolutely essential in the areas of debugging, the addition of routine features to all versions of the system, distribution of corrections to users, and conversion of SNOBOL4 to operate under MULTICS and TSS.

Nonetheless help was not forthcoming, and much of the burden of clerical work—even to making tape copies—fell to the developers themselves.

Certainly the attitude of BTL toward SNOBOL4 was reflected in its support of the project. Although SNOBOL4 was in considerable use within BTL and had received much favorable outside reaction, BTL's attention was directed toward the many important and pressing Bell System projects. SNOBOL4 could at best be considered peripheral. Because of its success, it was tolerated. The persons involved were allowed to work without significant hindrance. The lack of a formal project structure was mainly beneficial. Project members communicated informally but frequently. The notorious "11:15 lunch table" was the site of many exciting technical (and not-so-technical) discussions. The absence of a formal reporting structure, committee meetings, regular reports, and so on permitted the small, closely knit group to operate efficiently and effectively.

The local attitude toward research was exemplified by the "sand box" philosophy of our director, Bill Keister, who once told me that if you gave talented persons good facilities and left them alone, occasionally something good would happen (Keister, 1970). Clearly he placed SNOBOL in this category. There was no interference with our design nor were any features mandated by the needs of other projects at BTL.

### 1.13. The Release of SNOBOL

With the enthusiastic reception of SNOBOL within BTL, we were anxious to release it to outside users. Most software is developed with the goal of some sort of distribution to the computing community. In academic institutions, distribution of software is generally the concern and responsibility of the authors. Commercial ventures are predicated on distribution. SNOBOL, however, was developed within an industrial research organization and the original goal of the creators was to produce a tool for research going on there.

In 1963 there was little awareness within the computing community of the value of soft-

ware. Most of the thorny issues regarding the protection of investments and proprietary rights had not been addressed. While BTL had mechanisms for reviewing and approving release of internal information outside the company, these procedures were designed for talks and papers. The specific question of the release of computer programs had not yet been addressed, nor was the management of the company then really aware of the significance of software.

Since we were anxious to release the SNOBOL system to installations outside BTL, this situation posed a problem. Clearly making an issue or test case of SNOBOL to get a corporate decision and policy would not speed SNOBOL's release. Dave Farber took the position that, in the absence of a policy that prevented release of a program, approval to give a talk or paper on the program constituted release of that program as well.

Using this basis, the first release of SNOBOL was obtained by release of the original SNOBOL paper (Farber *et al.*, 1963a) for "publication in SHARE Secretary's Distribution SSD" (Renne, 1963). The SNOBOL system subsequently was available to IBM 7090/94 users through SHARE.

Most of the interest in SNOBOL was in universities. With BTL's traditional identification with academic institutions, there was considerable management support for the distribution of SNOBOL outside BTL. However, as more and more of the work in BTL shifted from hardware to software there was increasing corporate concern with protection of the company's investment in software and its proprietary rights in this area. Early attempts to provide this protection centered on patents. Patentability became an issue for SNOBOL with the request for publication of an internal memorandum on the SNOBOL pattern-matching algorithm (Griswold and Polonsky, 1963a). Release of the paper was held up for nearly a year before BTL decided not to attempt to patent the algorithm (Renne, 1964).

Approval for release of SNOBOL3 outside BTL was granted in October 1964 with the comment (Keister, 1964):

> Since SNOBOL is a computer language which solves no particular problems directly and discloses no proprietary information, it is agreed that we could release information on recent modifications to this language. This can consist of user's manuals and program listings in appropriate form. We should use our judgement in individual cases as to whether the distribution of this information is to the benefit of the Laboratories and will contribute to advances in the programming art. The Patent Department should be kept informed of the institutions and individuals to whom we are releasing this information.

This process was followed and the Patent Department was notified periodically as copies of the SNOBOL3 system and its documentation were distributed.

Meanwhile the Patent Department continued to work on the patentability of computer programs, with SNOBOL3 as one of its specific test cases. We felt that it was in the best interests of BTL and the computing community for SNOBOL3 to be freely distributed. We therefore took a stand in opposition to the Patent Department. Many discussions followed and devolved into the intricacies of patent law. We were told that Euclid's algorithm could have been patented if it had not, unfortunately, been previously disclosed. Our sarcastic suggestion of a patent disclosure in the form of the initial state of core storage just prior to execution was considered to be reasonable (Kersey, 1967).

Our management supported our position on the value of the public release of SNOBOL3. Eventually the Patent Department capitulated with the statement (Hamlin, 1965):

> In view of the public disclosure of the SNOBOL program in 1963, this subject matter does not

> appear to contain sufficient patentable novelty at this time to warrant the filing of an application for Letters Patent thereon.

Thus the first release of the original SNOBOL paper (Renne, 1963), which resulted in placing SNOBOL in the public domain, became the ultimate basis for public release of all subsequent material related to SNOBOL.

Release of SNOBOL4 outside BTL was of concern early in 1966 and release was obtained in May with the comment (Poage, 1966a):

> In the attached letter . . . dated October 20, 1964, principles were established for releasing information concerning SNOBOL outside of Bell Laboratories . . . . SNOBOL4, although different in syntax and implementation from SNOBOL, was not sufficiently new to consider for patent application. Therefore . . . the principles established in the attached letter should apply also to information concerning SNOBOL4. Further, since some economic benefit might be derived from the completed SNOBOL4 system, it was agreed that the final version should be submitted for copyright.

Once again the original release of SNOBOL provided the basis for releasing a new language. The phrase "final version" deserves note, since SNOBOL4 never reached a final version as such.

The issue was not completely settled until 1973 (Hirsch, 1973):

> SNOBOL, in its various forms, has been one of the most widely distributed of the Bell Laboratories–developed programs. Of the several versions of SNOBOL, the most current are designated as SNOBOL4 and SNOBOL4B. It is understood that the distribution of SNOBOL over the years has resulted in useful feedback and generally favorable reactions from those in the computing community at large.
>
> Because of the considerable administrative burden in reviewing individual requests for SNOBOL4 and SNOBOL4B, and because of the large number of copies already available to others, the Patent Division, with the concurrence of the Technical Relations Division, hereby grants a blanket approval for requests for these current versions of SNOBOL. Accordingly, any present or future requests for SNOBOL or its documentation need not be referred to the Patent or Technical Relations Divisions for approval. . . .
>
> The present grant of release approval is limited to the SNOBOL program (through versions 4 and 4B), and should not be considered applicable to other cases.

## 2. Rationale for the Content of the Language

### 2.1. Objectives

While the initial impetus for the development of SNOBOL was the need for a tool for formula manipulation, by the time the design of the language was under way, areas of interest had been extended to include graph processing and a number of text manipulation problems. Basically the objective was to provide a general-purpose language for manipulation of nonnumerical scientific data, such as formulas, that could be naturally represented by strings of characters. The interpretation of this objective was strongly influenced by the fact that we were oriented toward use of computers in research. An important criterion used in the design process was the "naïve user." We deliberately tried to produce a language that would be suitable for the person who was not primarily a pro-

grammer. We expected programs to be relatively small. In fact, the first implementation limited program size to 2000 "elements" (operators, operands, and so forth).

A main philosophical view emerged in the early design efforts: *ease of use.* To us, this implied several design criteria:

1. Conciseness: the language should have a small vocabulary and express high-level operations with a minimum of verbiage.
2. Simplicity: the language should be easy to learn and understand.
3. Problem orientation: the language facilities should be phrased in terms of the operations needed to solve problems, not the idiosyncrasies of computers.
4. Flexibility: users should be able to specify desired operations, even if these operations are difficult to implement. In particular, compile-time constraints should be minimized to allow maximum run-time freedom.

These objectives had several consequences, most of which can be characterized as a disregard for implementation problems and efficiency. This was a conscious decision and was justified by our contention that human resources were more precious than machine resources, especially in the research applications where SNOBOL was expected to be used. Our criterion for evaluating a proposed feature was more how it "felt" to us as programmers than how it conformed to conventional design criteria.

Specific consequences of this approach in SNOBOL were a single data type and the absence of declarations. In order to accommodate numerical computations in SNOBOL, arithmetic was performed (at least conceptually) on strings of digit characters that represented integers. This feature did not require declarations, and error checking was performed at run time. Declarations were viewed primarily as an aid to the implementation (Griswold, 1972a, p. 44), allowing compile-time decisions and the generation of efficient code. Since SNOBOL had only a single data type, the string, the most obvious declaration would have been for string length, either a fixed value or a maximum. However, the design criteria led us to support varying length strings with no specification by the user, placing all the burden of supporting this feature on the implementation. Similarly, SNOBOL contained no source-language storage management operations, again requiring the implementation to support allocation and garbage collection automatically.

From a language design viewpoint, such features amount to a design philosophy that the user of the language should not have to worry about how the computation is performed or what the actual implementation details are, but instead should be able to concentrate on the solution of problems. Without question, this philosophy was carried to the extreme in SNOBOL: deliberately and consciously so. It was the beginning of an attitude which I have sometimes described as "orthogonality to the mainstream."

### 2.2. Influence of Other Languages

One of the most fundamental problems we faced in the design of SNOBOL was determining the basic string manipulation operations and designing a notation for them. While algebraic languages had the well-established notation of mathematics as a natural basis for the elements of notation, no such independent notation existed for string manipulations. In developing the basic operations and notation, we were strongly influenced by earlier languages in this area.

Ivan and I were most familiar with SCL while Dave was more knowledgeable about COMIT. We had access to IPL-V, but we knew of other potentially relevant programming languages such as LISP only vicariously.

Since no information about SCL was ever released outside BTL, it is natural for observers to credit COMIT as the progenitor of SNOBOL. To us, however, SCL was a much more immediate influencing factor. The following quotation from the introductory section of the SCL manual (Lee *et al.*, 1962) characterizes its intent:

> The Symbolic Communication Language (SCL) is a language designed primarily for the processing of symbolic expressions. The basic unit which SCL deals with most conveniently is a line. A line may be an equation, an algebraic expression, a sentence, a set of pairs, etc. In mathematics, one often begins with a line, say an equation, and derives from it many other lines. Some of these derived lines are more important than others and are usually assigned a number. The general format in SCL is very much the same. The commands in SCL permit one to modify a line; the resulting line then appears next to the original line. If the new line is one which should be marked, a name can be assigned to that line. In this way it becomes particularly convenient to deal with mathematical expressions in SCL.

Since SCL had a strong influence on SNOBOL, it is appropriate to consider the factors that influenced SCL. Unfortunately, the only available source of information is the SCL manual. This manual references much of the prior literature on symbol manipulation languages, machine learning, and language translation. There is mention of IPL, LISP, COMIT, and macro processors, but no indication of how any of these might have influenced the design of SCL.

Aside from very different syntaxes, there is one clear difference between COMIT and SCL. While COMIT was oriented toward language translation with "constituent" units that typically were words, SCL was oriented toward algebraic manipulations on parenthesis-balanced strings of characters. While both SCL and COMIT had pattern matching and string transformation facilities, COMIT's facilities were constituent-based, while those of SCL demanded balanced parentheses. Both COMIT and SCL were organized around the concept of a workspace on which operations were performed. Other data was contained in "shelves" in COMIT and "data buffers" in SCL. Both SCL and COMIT were essentially single data type languages in which numerical computation was awkward. In COMIT, (integer) arithmetic was managed through operations on constituent subscripts. SCL, on the other hand, performed arithmetic only on rational numbers represented in prefix form as strings of character digits.

While it is difficult even for the designers of a language to recognize the influences of systems with which they are familiar, it was certainly the case that both SCL and COMIT were well known to us and we considered both to be inadequate for our needs. The resulting design of SNOBOL is certainly a synthesis of some features from both languages along with the introduction of new ideas. The extent of influence of SCL and COMIT is shown by that fact that SNOBOL contains their properties of basic string orientation, pattern matching and string transformation, and limited arithmetic facilities. SNOBOL departs by introducing the complete string (as opposed to a workspace of characters) as a data object that can be assigned to a variable. A more significant difference to users was that SNOBOL was considerably simpler than either COMIT or SCL, yet it provided more facilities for most text manipulation problems. Both SCL and COMIT were relatively slow and inefficient in their use of storage. Since we were confident that we easily could implement a

language that made considerably more economical use of resources, efficiency and storage utilization were not serious concerns. Similarly, neither SCL nor COMIT permitted really large programs, so we did not think in these terms in the design of SNOBOL. Therefore SCL and COMIT had the indirect influence of causing us, for the most part, to disregard questions of efficiency and the possible consequences of large SNOBOL programs.

The influence of machine language is shown in the indirect referencing operator, which elevated the apparently low-level operation of indirect addressing to a high-level language and allowed the value of an identifier to be used as a variable. This operation proved to be extremely important, since it provided a method for representing associative relationships among strings and hence for representing structures in a pure string context. Since indirect referencing could be applied to any string constructed at run time, it implied the existence of an internal run-time symbol table.

### 2.3. Basic Decisions

The choice of syntax for SNOBOL was closely related to the selections of its control structures and was primarily motivated by the desire to make SNOBOL convenient for the "naïve user." In the choice of syntax, the influence of COMIT is evident. The statement format, in which an operation was specified, resembles that of COMIT. However, in SNOBOL the subject specifies the string on which the operation is performed, while in COMIT the string is implicit, namely the current workspace. The selection of the next statement to be executed, conditioned on the success or failure of the operation, is found both in COMIT and in SCL. In SCL, however, it resembles the multiple-exit operations of machine order structures of the time. The decision to rely on conditional gotos instead of introducing more sophisticated control structures was a deliberate one. We felt that providing a statement with a test-and-goto format would allow division of program logic into simple units and that inexperienced users would find a more technical, highly structured language difficult to learn and to use. The decision not to introduce reserved words or something similar was motivated by the same concern. The initial enthusiastic response to the simplicity of the language reinforced this decision for future development. Changing to a more "modern" syntax was not seriously considered, even for SNOBOL4.

At the time of the development of SNOBOL, its free-form syntax with the absence of fixed field positions and lack of limitations on the length of identifiers was somewhat of a novelty. Again, these choices were deliberate and were designed to make SNOBOL easy to use. Users of SNOBOL, accustomed to languages with more rigid syntaxes, greeted the new freedom enthusiastically.

The positional syntax of the statement with a subject followed by an operation was motivated by the feeling that the subject as a focus of attention was a useful construct and would aid users in organizing their approach to writing programs. The importance of this view is described by Galler and Perlis (1970, p. 78).

The most controversial aspect of the syntax of SNOBOL was its use of blanks. The decision not to have an explicit operator for concatenation, but instead just to use blanks to separate the strings to be concatenated, was motivated by a belief in the fundamental role of concatenation in a string-manipulation language. The model for this feature was simply the convention used in natural languages to separate words by blanks. Furthermore, an explicit operator for concatenation was thought to be overbearing, especially considering the frequency of the appearance of concatenation in SNOBOL programs.

(The consequence of the ambiguous use of blanks, such as to separate subjects from the operations performed on them, is discussed in Section 3.3.1.)

One important early decision was to include backtracking in SNOBOL pattern matching. COMIT did not include backtracking, so that once a constituent matched, no alternative match was attempted, even if a subsequent constituent failed to match. Yngve justified this decision on the basis that someone had shown a pattern that would take 300 years of computer time to complete if backtracking were allowed (Yngve, 1964). We had observed that COMIT pattern matches sometimes failed when they should have succeeded, at least intuitively. We felt that it was conceptually important for a pattern to be a specification of the structural properties of a string and that all alternatives should be attempted to fit the pattern to these structural properties. The theoretical possibility of pattern matching taking arbitrarily long did not seem as important to us as the practical issues. In any event, we did not believe that this problem would arise in practice although steps were taken to minimize the problem (see Section 3.3.3). I know of only one instance where a pattern match actually took so long that it was mistaken for an endless loop within the system.

It is interesting that the issue of character sets was never given much consideration, despite SNOBOL's emphasis on string manipulation. The 48-character BCD character set was in general use when SNOBOL was designed, and it was accepted without much thought. Even with SNOBOL4, where various implementations were undertaken on computers with characters sets of different sizes, orderings, and graphics, no design consideration was given to the problem. Instead, it was considered to be an implementation problem specific to the target computer.

Machine independence and portability were given no consideration in the original design of SNOBOL, although use of string-manipulation macros (McIlroy, 1962) in the implementation laid the basis for later work. Our horizons were limited and our concerns were for providing a string manipulation language for the local computer, an IBM 7090. At that time, all BTL computer centers had 7090s as their main computers, and the possibility of implementations for other computers had not yet risen.

### 2.4. Progression to SNOBOL2 and SNOBOL3

Users quickly outgrew SNOBOL and the absence of certain facilities in SNOBOL made some programming tasks impractical or impossible. In one sense, SNOBOL was too simple. In another sense, the success of SNOBOL attracted users from areas in which the use of SNOBOL had not been anticipated.

An example of the deficiency of SNOBOL was the absence of any direct method for determining the length of a string. This computation had to be programmed using pattern matching. A naïve, although typical, method was to remove a character at a time until the string was exhausted, meanwhile incrementing a running count. Since the length of every string was readily accessible internally, this deficiency was doubly indefensible. There were no numerical comparison operations and these computations also had to be accomplished by pattern matching.

An obvious solution to these and a host of similar problems was the introduction of built-in functions to provide a repertoire of operations under a single syntactic cloak.

While the repertoire of built-in functions in SNOBOL2 handled the most commonly needed computations, special needs (such as interfacing operating system facilities) could

not be accommodated by any fixed set of built-in functions. This motivated the introduction of an external function mechanism (Farber *et al.*, 1965b) whereby users could augment SNOBOL2 by functions coded in assembly language. This facility was used extensively (Manacher, 1964; Manacher and Varian, 1964; Griswold and Varian, 1964; Flannery, 1964; Griswold and Polonksy, 1965, Griswold, 1965, 1966a; Calise, 1966; Hall and McCabe, 1967; Wilson, 1967).

Like SNOBOL, SNOBOL2 lacked any mechanism for allowing the programmer to extend the operation repertoire at the source level. More serious, perhaps, was the fact that there was not even a mechanism for programming subroutines at the source level. While a subroutine linkage mechanism could be simulated using indirect referencing and computed gotos, this technique was awkward and was not obvious to the novice. By modern standards, most SNOBOL programs were extremely unstructured.

The introduction of a mechanism for defining functions distinguished SNOBOL3 from SNOBOL2. In this case, serious design problems arose. SNOBOL and SNOBOL2 had no block structure or mechanism for the declaration of the scope of identifiers, and all labels were global. We decided to retain this "simplicity" by adding a built-in function, DEFINE, which established the existence of programmer-defined functions at run time. The mechanism itself was simple. The bodies of defined functions were written as segments of a SNOBOL3 program. When a DEFINE was executed at run time, the entry point, function name, formal arguments, and local variables were bound. This mechanism avoided the need for declarations and provided considerable run-time flexibility, since the arguments of DEFINE, including even the name of the function, could be computed at run time. By saving and restoring the values associated with the formal parameters and local variables on call and return, respectively, a form of dynamic scoping was obtained and recursion worked properly.

In the design of the defined-function mechanism, the original SNOBOL design philosophy was clearly in conflict with contemporary concepts of program structure and the growing needs of users. The flexibility of run-time definition with the possibility of redefinition and computation of function attributes had some appeal for esoteric applications, but the lack of scope declarations and locality of labels virtually precluded well-structured programs and separately compilable modules. The mechanism chosen was partly the result of adherence to the orginal SNOBOL tradition and philosophy and partly the unwillingness to become involved in more substantive issues that an alternative approach would have forced. Another factor was limited personnel resources, which certainly inhibited radical departures from previous design. Even the chosen design for defined functions was set aside until Lee Varian became available on a summer assignment. Since Dave, Ivan, and I were doing essentially all of the implementation, distribution, documentation, and maintenance, other language features that would have required a substantial implementation effort were virtually unthinkable.

### 2.5. SNOBOL4

Despite the addition of the function mechanisms, SNOBOL3 retained most of the original SNOBOL design: a single string data type, static specification of patterns with a limited repertoire of pattern types, and indirect referencing as the only mechanism for representing structural relationships.

Work on structural manipulation functions (Griswold and Polonsky 1965 and Griswold

1965) had shown the usefulness of including facilities for processing data structures in a string-based language. Other inadequacies of SNOBOL3 were also apparent, notably the static, limited range of pattern specification and particularly the absence of a mechanism for specifying alternative matches.

The solutions to these problems were not so evident. While there were many suggestions, including a multitude of specific data structures and a variety of syntactic elaborations for pattern specifications, the unification needed for good language design was lacking. For my part, I resisted further embellishments to SNOBOL3. In addition, SNOBOL3 generally satisfied most of the users' needs and there was not the pressure to go to a new language that there had been in the transition from SNOBOL to SNOBOL2.

As mentioned in Section 1.8, the pressure eventually came from another source—the impending transition to third-generation hardware. The possibility of new computers compounded our earlier concerns with implementations of SNOBOL and SNOBOL3 done outside BTL. These independent implementations had proved difficult because of lack of previous consideration of machine independence and because the only implementation information was the assembly-language listings of the programs. We began to see the basis for a portable implementation in the extension of McIlroy's string manipulation macros to a complete set of machine-independent assembly-language operations.

During this period, the idea came that satisfied my reservations about the design of a new language—the concept of patterns as data objects that could be built, combined, and manipulated dynamically as opposed to the static pattern specification of SNOBOL3.

While SNOBOL introduced a new, simple string language and SNOBOL2 and SNOBOL3 fleshed out this design, SNOBOL4 departed radically from earlier work. Much of the simplicity of SNOBOL3 was abandoned for greater range and more powerful features. The semantic content of SNOBOL4 was substantially greater than that of SNOBOL3. While much of the early philosophy of flexibility was expanded, SNOBOL4 became a general-purpose language where SNOBOL3 had been a string manipulation language. The rationale for this change is found in the evolving sophistication of both the users and the designers. In a sense, it was an inevitable consequence of professional development and the interest in exploring new issues.

The issue of compatibility with previous languages was first seriously raised in the design of SNOBOL4. Except for a few minor syntactic changes, SNOBOL2 and SNOBOL3 were upward compatible with SNOBOL. There was concern that substantial changes in syntax for SNOBOL4 would make SNOBOL3 programs obsolete. We felt, however, that in order to realize the potential of new ideas, substantial changes were needed and that it would be best to make a clean break, regardless of the impact on SNOBOL3 programmers. The attempt to maintain upward compatibility in the progression from COMIT to COMIT II (Yngve, 1972) provided us with an example of the inhibiting effects of the alternative choice.

However, features of SNOBOL3, except those specifically marked for redesign, were not critically examined for modification or deletion. For example, indirect referencing was included in SNOBOL4 without much thought, despite the fact that its primary justification (representation of structural relationships) no longer applied with the introduction of arrays and defined data objects in SNOBOL4. Similarly, the lack of declarations persisted, even though the variety of data types that had been introduced would have made type declarations meaningful. Defined functions in the style of SNOBOL3 were also retained despite the otherwise radical departure of SNOBOL4 from SNOBOL3.

The computing environment at BTL until this time had been entirely batch oriented and we had not given any consideration to operation in an interactive environment, although there was beginning to be local use of outside time-sharing services. Since MULTICS was slated to replace the batch operation with a BTL-wide time-sharing facility, language features for interactive computing might have been given major consideration. I recall, however, that we gave relatively little attention to this issue. We were influenced by the opinion of local users of time sharing who placed more importance on run-time capabilities than on interactive compiling (Sinowitz, 1966).

The main attributes of SNOBOL4 that deserve mention are:

1. The multitude of data types.
2. Patterns as data objects with a large variety of pattern types.
3. Data structures, including arrays, tables, and defined (record-type) objects.
4. Run-time compilation.
5. Unevaluated expressions.

The need for more than one data type was a consequence of patterns as data objects. The inclusion of other data types followed naturally, but slowly. The gradual addition of data types characterizes the evolutionary development of the design of SNOBOL4. SNOBOL4 started with three data types: strings, integers, and patterns. In late 1966, arrays and defined data objects were added. Real numbers, code, and names were added in 1967, unevaluated expressions were added in mid-1968, and tables in mid-1969.

The introduction of data types such as integers and real numbers raised the issue of coercion, the automatic conversion of the data type to suit the context. We viewed coercion together with the absence of declarations as an important aspect of ease of use. Thus a string representing an integer could be read in and used in arithmetic contexts without specification on the part of the programmer. Mixed-mode arithmetic and other polymorphic operations followed naturally from this philosophy.

Patterns as data objects represented a major design change. Previously each type of pattern was represented by a different syntactic notation and each pattern had to be explicitly written in line at the place where it was used. The addition of new pattern types in this scheme would have required a proliferation of syntax. Furthermore, the requirement for statically specified, in-line patterns placed practical limitations on their complexity. With patterns as data objects, however, an unlimited number of pattern types was possible without additional syntax. Built-in patterns were included as the initial values of selected identifiers and a repertoire of pattern-valued functions was provided as a means of constructing more complex patterns. Thus patterns could be built incrementally, reducing the complexity needed at each step, and the same pattern could be used wherever needed without the user having to duplicate its construction.

Data structures were significant in providing a natural method of organizing data. In the spirit of SNOBOL, SNOBOL4 arrays were designed so that they were created at run time with dimensionality and size that could be computed. The decision to treat arrays as objects with a distinct data type allowed passing arrays as values and treating them like other program objects. Again in the spirit of SNOBOL, there were no type declarations for arrays and hence they were heterogeneous.

Defined, record-type data objects were added to provide a mechanism whereby the user could create data structures. This method was motivated by the opinion that no number of

specific built-in data types such as lists, trees, and stacks, could satisfy all user needs. Thus the philosophy was to provide a means whereby the user could construct any kind of data structure and define operations on it.

Tables, which provided a kind of associative data structure, had been suggested a number of times, notably by Doug McIlroy and Mike Shapiro. It was Doug's persistence that resulted in their addition to SNOBOL4 in mid-1969, at a very late stage in the development of SNOBOL4 and at a time when there were very few remaining personnel resources for additional changes. Doug commented, as I recall, that tables seemed like a natural thing to have in a language such as SNOBOL4, since many of the applications of SNOBOL4 involved the need for operating on symbol tables at the source-language level. This one instance of outside influence, which proved to be very beneficial, stands out in my mind, since most other issues of language design were raised and decided within the project group.

The strong influence of a flexible, machine-independent implementation is shown in run-time compilation and unevaluated expressions. The interest in run-time compilation dates back to the influence of SCL, which not only had run-time compilation, but in fact routinely required its use. SNOBOL was designed with the idea of run-time compilation, in which labels were treated as variables that had their associated statements as values. The idea was that if a new value was assigned to a variable that occurred as a label, this new value would be considered as a string representing a statement and consequently compiled. While the treatment of labels as variables was supported, compilation of newly assigned values was not. As a result, if a label was accidentally used as a variable in a program, subsequent execution of the statement corresponding to that label produced the error message "RECOMPILE NOT ENABLED" (which soon became infamous).

SNOBOL2 and SNOBOL3 did not attempt run-time compilation, but the idea was not forgotten. In the design of SNOBOL4, the issue of run-time compilation was again raised, but a method of handling it still was not evident (the SNOBOL method was clearly unsatisfactory, since an accidental use of a label as a variable, possibly depending on data, could destroy a program). It was the implementation of SNOBOL4 that eventually suggested the design of run-time compilation.

To provide machine independence, portability, and flexibility, the implementation was designed around the generalization of McIlroy's string macros as a hypothetical assembly language for an abstract machine (Griswold, 1972a, 1976a). To achieve uniformity, all data were represented in terms of *descriptors*, the "word" in this abstract machine. Each SNOBOL4 data object was represented by a descriptor, either containing the datum or a pointer to it. An array, for example, consisted of a descriptor pointing to a block of descriptors, each of which represented an element of the array. Code compiled for a SNOBOL4 program also consisted of a block of descriptors. The idea for run-time compilation came as a result of the observation that a block of code could be treated as a source-language data object merely by having a descriptor point to it. It was a simple matter to add a function that would call the compiler at run time to convert a string representing a sequence of SNOBOL4 statements into a block of code, thus augmenting the original source-language program. In fact, it took just two days from the inception of the idea until run-time compilation was available in SNOBOL4.

An essential component for the practicality of this feature was the availability of the compiler during program execution. This was the consequence of environmental factors

and the desire for a portable implementation. In the first place, the compiler was small (since SNOBOL4 has few compile-time operations and the interpretive implementation avoided the need for a bulky code generator or optimizer). As a result, there was not a great motivation to segment the implementation into a compilation component followed by an execution component. In the second place, the desire for a portable system discouraged an organization that required linkage between separate segments, overlays, and the like. Thus the compiler was just "there," which turned out not only to be handy, but also actually to make run-time compilation conceivable.

One concern from the initial SNOBOL design was dynamic components in patterns. The need for such a feature is typified by a specification such as locating two occurrences of the same character in a string. In SNOBOL this was handled in a rather ad hoc fashion by "back-referencing" identifiers that had values assigned earlier in a match. A more general problem, however, lies in recursive definitions such as those found in production grammars for languages.

The solution to these problems in SNOBOL4 first came in the form of "deferred evaluation," in which a variable could be flagged so that its value was not fetched until it was encountered during pattern matching. Thus a pattern could contain a deferred reference to itself, producing the effect of a recursive definition. While this scheme solved some of the problems related to the need for dynamic evaluation during pattern matching, it did not solve the general problem.

Again the implementation suggested a better mechanism. Since expressions were compiled into sequences of descriptors within code blocks, a descriptor pointing to the code for an expression could also be a source-language data object. As a result any expression could be left "unevaluated" by providing an operator that simply returned a pointer to the appropriate position in the code and then skipped over the code instead of executing it. Subsequent evaluation of the code could then be accomplished by temporarily changing the position for execution to the previously unevaluated code. When appearing in patterns, such unevaluated expressions were evaluated during pattern matching, which provided the same facility as deferred evaluation. Since any expression could be left unevaluated, however, there was the possibility for performing any arbitrary computation during pattern matching. With this feature, all the theoretical limitations of pattern matching were removed. Not only was there a natural isomorphism between production grammars and SNOBOL4 patterns (Griswold, 1975a, pp. 7–12, 233–234), but context sensitivity and Turing grammars could be represented within SNOBOL4 patterns.

Experimentation was one consequence of the evolutionary development of SNOBOL4 and especially the continuing design with the availability of a flexible implementation. It was easy to try out design ideas in practice and to modify or discard them on the basis of actual use. While there were certainly ideas that were tried and discarded, most experimental features were, in fact, retained in some form. This method of language development was largely responsible for the large semantic size of SNOBOL4 compared with the earlier SNOBOL languages

Most features suggested for SNOBOL4 were implemented. Among suggestions that were not implemented were some cosmetic improvements, such as a program cross-reference listing, which was implemented experimentally but was never incorporated into the official version, and structure dumps, which have since been incorporated, after a fashion, in SPITBOL (Dewar, 1971) and SITBOL (Gimpel, 1974c).

### 2.6. Language Definition

Definition of the SNOBOL languages was always treated informally. The earliest documentation of SNOBOL included a BNF grammar describing the syntax, but the semantics were dealt with in an informal manner (Griswold, 1963a). The first published paper on SNOBOL treated the syntax informally as well (Farber *et al.*, 1964a). This informality continued with SNOBOL2 and SNOBOL3 (Farber *et al.*, 1966).

With SNOBOL4, more care was given to the definition of the syntax. Here the "LSD" notation originally used by IBM for describing the syntax for PL/I was applied (IBM, 1970, pp. 241–242). Since SNOBOL4 itself provides a notation for describing syntax, it is not surprising that SNOBOL4 has been used to describe its own syntax (Griswold, 1966; Griswold *et al.*, Polonsky, 1971, pp. 226–229). The idea of enriching BNF-style syntax description systems by adding constructions in the style of SNOBOL4 appears in Griswold (1972a, pp. 261–264).

Language semantics were always treated casually by the designers of SNOBOL. It has been a long-standing joke that the only real description of the formal semantics of SNOBOL4 is contained in the program listing of the SIL implementation (Griswold, 1976b). Despite the difficulties posed by SNOBOL4's dynamic characteristics, work has been done by others on more formal approaches to its semantics (Herriot, 1973a,b, 1974; Mickel, 1973; Tennent, 1973; Stewart, 1975).

### 2.7. Standardization

Until SNOBOL4, there was no serious consideration of standardization. In fact there was a conscious decision to discourage both standardization and control by user groups. However, dialects and variant implementations of SNOBOL3 done outside our group were posing significant problems for users and the question of standardization for SNOBOL4 was considered. As a matter of philosophy, we felt that SNOBOL was a living language and that change and experimentation were more beneficial than constraint and standardization. The machine-readable source was freely distributed to encourage experimentation. In fact, standardization would have been impractical in any event, since there was no mechanism to control or prevent the development of independent implementations or language variants.

However, the SIL implementation imposed de facto standardization on SNOBOL4 that exceeded most of the deliberate standardization attempts. SIL implementations of SNOBOL4 are more nearly compatible than those of most other generally available programming languages.

## 3. A Posteriori Evaluation

### 3.1. Meeting of Objectives

With the evolution of SNOBOL, SNOBOL2, SNOBOL3, and SNOBOL4, the original objective of developing a tool for string manipulation gradually broadened to include a wider range of non-numeric applications. The desire to provide a tool for our own research became more a desire to provide a tool for a large community of users and to make contributions to the design and implementation of programming languages.

In retrospect, the original objectives were greatly exceeded, but the areas of application for SNOBOL4 (the only one of the SNOBOL languages now in significant use) are substantially different from those originally anticipated. For example, during the development of SNOBOL, the designers gave no thought to potential uses for document formatting or for computing in the humanities, although these have become some of the most significant applications of SNOBOL4 (Griswold, 1977b). While SNOBOL has had some applications in formula manipulation (Bailey *et al.*, 1969; Rodgers, 1966; Roosen-Runge, 1967), the problem that provided the initial motivation for the development of the new language, it has never enjoyed significant popularity in this area. Similarly, use of SNOBOL as a compiler-writing tool was envisioned (Farber, 1963a), but the suitability of SNOBOL4 in this area is at best controversial (Dunn, 1973; Hanson, 1973; Gimpel, 1976, pp. 406, 411–432; Wetherell, 1978, p. 146). In fact SNOBOL4 is not now widely used for implementing production compilers.

For the areas in which SNOBOL4 has proved useful, the users generally like SNOBOL4 and feel that it meets its objectives (as they perceive them). SNOBOL4, like its predecessors, is the kind of language that has its adherents and ardent supporters (as well as its detractors). It is similar to APL in this respect, although the feelings about SNOBOL4 generally are not as strong as those about APL.

It is interesting to note that not all users greeted the increased capacity of SNOBOL2, SNOBOL3, and SNOBOL4 with enthusiasm. Some preferred the simplicity of the original SNOBOL, and SNOBOL4 was resisted by some experienced programmers despite the additional power it provided (Uhr, 1968; McIlroy, 1977).

The objectives of the designers of the SNOBOL languages have not always been well understood. In my opinion, the computing community as a whole has little accurate knowledge of SNOBOL and treats it more by hearsay than by actual knowledge. While there is a substantial amount of available material on the SNOBOL languages (Griswold, 1977b), effort is required to extract the real attributes of SNOBOL4. In general, one must use SNOBOL4 to understand it; it does not come naturally as yet another "algebraic" language might to a FORTRAN or PL/I user. Similarly, many persons are unaware of the substantial differences between SNOBOL and SNOBOL4. Here a decision to go to an entirely new name might have avoided the assumption that all the SNOBOL languages were similar.

User acceptance has been strongly influenced by the ready availability of the SNOBOL languages. From the earliest development, SNOBOL was distributed without charge or restriction of use to anyone interested in it. Its source code was provided without restriction, which facilitated its installation. For the most part, documentation and program material were distributed by BTL without charge. An interested person had only to supply a tape. Thus individuals and organizations with limited financial resources were able to get the programs. With SNOBOL4 the portable implementation was made freely available, and technical support was provided for implementation on a variety of machines. The exchange of program material for the portable implementation of SNOBOL4 in return for effort freely given by implementors on other machines was essential, especially in the absence of vendor support or a commercial base. It is my opinion that this free distribution and open attitude toward release of information was, in fact, *essential* to the acceptance and success of SNOBOL. If SNOBOL had been protected, if its distribution had been controlled, or if a charge had been made for the program material, it is likely that SNOBOL would have remained one of the many interesting, but noninfluential ideas of com-

puter science. The use and influence of COMIT, for example, was probably substantially diminished by tight control over release of the source program. For an interesting debate of the issue of unrestricted distribution, see Galler (1968) and Mooers (1968).

### 3.2. Contributions of the Languages

SNOBOL is responsible, primarily, for introducing to high-level languages the idea of the string of characters as a single data object rather than as a workspace or an array of characters. This is a conceptual matter, and one of considerable importance, since the treatment of a varying length string as a single data object permits a global view of string processing that character aggregates inhibit.

One of the most formidable design problems faced by the architects of the SNOBOL languages, particularly at the beginning, was the implementation of storage management for a large number of varying length strings. However, the freedom that string data objects provide the programmer is enormous. Compare the 127 shelves of COMIT and the 64 possible string variables and two data buffers of SCL to SNOBOL, where strings were simply data objects in the sense that integers are data objects in FORTRAN.

SNOBOL's most obvious specific contribution is a viable high-level approach to string analysis through pattern matching. Where COMIT and SCL introduced pattern-matching features that were intriguing but of limited practical use, SNOBOL, and SNOBOL4 in particular, have made this tool generally well known and widely used.

More personally, I view the best points of SNOBOL to be the ease of use and its "comfortable feeling." It is somewhat like an old shoe. SNOBOL programmers characterize it as a hospitable, friendly language that allows much to be accomplished quickly and easily. Its "one-line" programs for problems such as character-code conversion and reformatting data cards are famous.

Some of the esoteric features in SNOBOL4 have proved valuable and have allowed comparatively easy treatment of otherwise intractable problems. Run-time compilation, for example, permits SNOBOL4 code to be imbedded in data, thus allowing easy run-time extension of programs such as document processors (Griswold, 1975a, pp. 189–191, 270; Anderson and Griswold, 1976; Conrow, 1977). Similarly, the capability to redefine any built-in SNOBOL4 operator allows extensions and experimental simulation of more advanced language features (Griswold, 1975a, pp. 26–30, 115, 256–257). This capability was used by John Doyle to simulate extensions to pattern matching in SNOBOL4 (Doyle, 1975).

As SNOBOL progressed to SNOBOL2, SNOBOL3, and then SNOBOL4, I applied one personal benchmark: could problems be solved by the new language that were not practical with the old one? In each case, the answer was yes. There are two aspects to this measure. In some cases, ease of programming made the solution feasible in terms of the effort or time required. As Kernighan and Plauger (1976, p. 84) remark, "many jobs will not get done at all unless they can be done quickly." In other cases, solutions to problems that were formerly intractable or unthinkable because of the degree of programming difficulty became reasonable and attractive. This latter situation applies particularly to researchers in areas such as the humanities, where sophisticated programming skills are less common.

*3.2.1. Other Implementations*

A measure of the importance of SNOBOL is given by the number of implementations carried on outside BTL. Many other persons and groups decided to implement various versions of SNOBOL. One of the earliest implementations was an "ALGOLized" version of SNOBOL for a Soviet computer (Kostalansky, 1967; Lavrov, 1968). Parallel to the development of SNOBOL2 and SNOBOL3, there were several implementations motivated both by the need for a SNOBOL language on machines other than the IBM 7090/94 and by an interest in experimentation with extensions and changes. The first implementations were completed for the RCA 601 (Simon and Walters, 1964), the PDP-6 (Guard, 1967), the Burroughs 5500, the CDC 3600, the SDS 930 (Lampson, 1966; Kain and Bailey, 1967), the IBM 360 (Madnick, 1966), and the IBM 1620 (Wilson, 1966). Many stories have been told about the 1620 implementation. It was a pure interpreter, constantly reprocessing the source program text. The call of a built-in function is said to have taken 40 seconds and programs were (naturally) frequently left to run overnight. Nonetheless, this implementation apparently was satisfactory and is remembered with some fondness by its users.

With SNOBOL4, there was even more implementation activity by other groups. This activity took two forms: implementations of the portable SIL system (which produced standard SNOBOL4 implementations) and independent implementations with various goals (which produced various dialects and language variants). The SIL system has attracted implementors for two reasons: the desire for SNOBOL4 on a specific computer for which it was not available, and the interest in the implementation of portable software in its own right.

As mentioned in Section 1.9, the first transporting of the SIL implementation of SNOBOL4 was done by Jim, Ivan, and myself, when we moved the IBM 7094 implementation to the IBM 360 in late 1966. In 1967 and 1968, SIL implementations were started for the CDC 6000 series (Gaines, 1968; Shapiro, 1969), the GE 635 (at the BTL Murray Hill Laboratory), the UNIVAC 1108 (Osterweil, 1970), the RCA Spectra 70, the CDC 3600, the Atlas 2 (Moody, 1972), the XDS Sigma 7 (Barker, 1973), the DEC-10 (Wade, 1970), and the Datacraft 6000 (Datacraft Corporation, 1972). All of these implementations were completed. In the process, much valuable feedback was obtained about the language and problems in the design and structure of the implementation. In all, SIL implementations have been undertaken for 40 different computers, and most have been brought to a working state. Recently even an implementation for the IMSAI 8080 has been started. In the case of some computers, several independent implementations have been done by individuals who wanted the experience of implementing portable software, even though another implementation was already available.

The relative inefficiency and large size of the SIL implementation of SNOBOL4 has challenged a number of groups to attempt more economical implementations. CAL SNOBOL approached the problem by severely subsetting SNOBOL4 to gain a dramatic performance improvement (Gaskins, 1970). SPITBOL produced the first efficient "compiler" system that implemented a reasonable approximation to the entire language, including runtime compilation (Dewar, 1971; Dewar *et al.*, 1975; Berndl, 1975). FASBOL combined subsetting with optional declarations to achieve very efficient implementations for the UNIVAC 1108 and the DEC-10 (Santos, 1971). SITBOL proved that an interpretive implementation could be a sufficiently efficient for production work (Gimpel, 1973b). SNO-

BAT combined some subsetting with new implementation techniques to produce a fast batch compiler (Silverston, 1976a,b, 1977). Most recently MACRO SPITBOL, a machine-independent and portable interpretive implementation has been developed (Dewar and McCann, 1977). This system performs better than some implementations tailored to specific machines. To date, MACRO SPITBOL has been implemented for the ICL 1900 (Dewar and McCann, 1974), the Burroughs B1700 (Morse, 1976), the DEC-10 (McCann *et al.*, 1976), the CDC 6000 series (Druseikis and Griswold, 1977), and the PDP-11, (Dewar, 1977).

In total there are some 50 implementations of SNOBOL4 currently available for different computers and operating systems (Griswold, 1978a).

### *3.2.2. Dialects and Extensions*

These implementations resulted in numerous dialects of the language. In FASBOL, Santos added optional declarations to permit the generation of efficient code. SPITBOL added a number of built-in functions and error recovery facilities that have proved useful. SITBOL included most of the SPITBOL enhancements and added others (Gimpel, 1973c, 1974c). SITBOL is also notable for its integration with the DEC-10 operating system environment, which makes it particularly pleasant to use.

Changes to the SNOBOL4 language made in these implementations divide them into two categories. Those with substantial differences (CAL SNOBOL, FASBOL, and SNOBAT) tend to be limited in their use because they cannot run many standard SNOBOL4 programs. On the other hand, SPITBOL, SITBOL, and MACRO SPITBOL are sufficiently compatible with the standard version of SNOBOL4 that they usually can be used to run existing programs with few if any modifications. It is interesting to note that the support of run-time compilation is an important issue here. A surprising number of "production" SNOBOL4 programs use this feature in an essential way (Griswold, 1975a, pp. 189–191, 222–224; 1975d; Anderson and Griswold, 1976; Gimpel, 1976, pp. 353–359; Conrow, 1977).

A number of significant extensions have been made to SNOBOL4. Notable are SNOBOL4B, a SIL-based extension that includes three-dimensional character "blocks" (Gimpel, 1972) and SNOBOL+, which adds pattern matching on trees (Ophir, 1974).

### *3.2.3. Influences on Hardware*

Chai and DiGiuseppe (1965) considered issues of hardware design to support SNOBOL-like features. M. D. Shapiro (1972a,b) designed a SNOBOL machine using conventional hardware. A number of attempts have been made to microprogram SNOBOL4 or SIL (Syrett, 1971; Rosin, 1969; Rossman and Jones, 1974). So far, such attempts have had only limited success. While the B5500 and CDC STAR computers have introduced interesting string operations, the influence of SNOBOL on machine order structures is apparently tangential.

### *3.2.4. Use of SNOBOL4 as an Experimental Tool*

As mentioned in Section 1.9, one of our motivations for developing SNOBOL4 was to have a vehicle for experimentation with programming language features. Once the project got under way, it became so overwhelming that there was little possibility for experimentation outside of the main thrust of the language. After the language reached a stable condition in 1970, some experimentation was undertaken at BTL with a SIL-based variant of

SNOBOL4 into which some speculative language features were incorporated (Griswold, 1971).

One experiment, while not speculative as a language feature, was interesting in its use of the run-time compilation facility. Users had long wanted a program cross-reference facility. Instead of implementing this directly in SIL, it was coded as a SNOBOL4 program that constructed cross-reference listings. A string corresponding to this program was then incorporated in the SIL data region. During program initialization, this string was compiled as a SNOBOL4 program segment to provide the desired facility. While the implementation of SNOBOL4 was not designed for bootstrapping, the source-language run-time compilation facility nevertheless supported this kind of process, which Jim Gimpel calls "spatial bootstrapping" (Gimpel, 1973d). Jim used a similar technique to provide a histogram facility for SITBOL (Gimpel, 1974c).

Concentrated experimental work started in 1971 at the University of Arizona to exploit SNOBOL4 as a research tool. Early work led to several additions to SNOBOL4 to extend its usefulness in this area (Griswold, 1975c). Subsequent applications led to new parameter transmission techniques (Druseikis and Griswold, 1973), high-level data structure processing facilities (Hallyburton, 1974), a linguistic interface to the SIL abstract machine (Griswold, 1975b) and novel performance measurement facilities (Ripley and Griswold, 1975).

### 3.3. Mistakes and Problems

#### *3.3.1. Syntax and Control Structure*

There are many things to criticize about any programming language and SNOBOL is no exception. The most controversial aspect of SNOBOL4 is its primitive syntax and lack of "modern" control structures. This is a debatable issue. SNOBOL has long been popular among nonprofessional and "novice" programmers who find the simple test-and-goto structure of SNOBOL to be much preferable to the nested control structures of languages that are more sophisticated in this respect. While the syntax and control structures of SNOBOL have been continually criticized by the more sophisticated component of the computing community, it may just be that this aspect of SNOBOL was primarily responsible for its initial acceptance and its practical usefulness.

At the present time, such a syntax and control structure would be "socially unacceptable" in the design of a new language. With the maturation of the programming community, the time for a SNOBOL-style syntax may be past; it certainly is not a necessity. Alternatives have been suggested (Griswold, 1974; Lindsay, 1975; Tamir, 1974) and SL5 provides an integration of the SNOBOL-style signaling mechanism with conventional control structures (Griswold and Hanson, 1977).

A less controversial problem is the defined function mechanism of SNOBOL4, which separates the defining statement from the procedure body, provides no static specification to delimit the procedure body, and does not provide scoping for labels. SNOBOL4 programmers "get around" these problems, but not happily (Abrahams, 1974). In retrospect, this aspect of SNOBOL4 seems to be an out-and-out mistake.

A few minor aspects of the syntax of SNOBOL have caused problems all out of proportion to their apparent significance. The use of blanks is the most notable of these problems. It is debatable whether the absence of a specific operator for concatenation was the

best choice. While it is convenient to write strings to be concatenated with only separating blanks, many programmers find that the resulting requirement for blanks to surround infix operators is a compensating inconvenience. Many program errors result from the failure to provide mandatory blanks and the absence of a concatenation operator makes programs harder to read. The dual use of blanks to separate labels and gotos from the rest of statements and to separate the subject from the pattern compounds the problem. If I were to do it over again, I would, at least, provide explicit operators for concatenation and pattern matching, although others disagree on this point (McIlroy, 1977).

The fact that assignment and pattern matching are statements, not expressions, is a curious anomaly that can be traced to the earliest design considerations where the subject as a focus of attention was held to be of primary importance. The burden of these restrictions is not as great as it would be if SNOBOL had more sophisticated control structures. Most programmers are not aware of the effects of these limitations on their programming styles. Later dialects such as SITBOL and MACRO SPITBOL have corrected these mistakes.

As might be expected, several preprocessors have been written to convert more conventional control structures into standard SNOBOL4 (Croff, 1974; Beyer, 1976; Hanson, 1977b; Sommerville, 1977).

### *3.3.2. Structures and Indirect Referencing*

SNOBOL, SNOBOL2, and SNOBOL3 were hampered by the single string data type, which made arithmetic awkward (and inefficient) and required programmers to simulate structures by encoding them as strings or producing awkward and "dangerous" pointer networks using indirect referencing. SNOBOL4 corrected these problems with a vengeance, introducing nine built-in data types and the ability to create the effect of others by using defined data objects.

Indirect referencing itself is hard to understand, error-prone, and frequently misused. While indirect referencing was valuable in SNOBOL, SNOBOL2, and SNOBOL3 in the absence of more direct ways to represent structural relationships, its continuance in SNOBOL4 was probably a mistake. With the exception of computed gotos, indirect referencing is unnecessary in SNOBOL4, since tables provide the same facility with the added capability to restrict the domain of applicable strings. By the time that tables were added, however, indirect referencing was ingrained in the language.

### *3.3.3. String Processing and Pattern Matching*

A curiosity of SNOBOL4 is its relative lack of primitive string operations. Since pattern matching is so powerful, it is possible to do virtually anything using it. The results may be bizarre and inefficient, however. For example, SNOBOL4 does not have a primitive substring operation and pattern matching is necessary to perform even this simple operation. While SPITBOL and SITBOL rectified this defect, even these dialects lack primitives for "lexical" operations such as locating the position of a substring within a string. SL5 shows that there is no conceptual barrier to such primitives (Griswold *et al.*, 1977).

While pattern matching is certainly the most powerful facility in SNOBOL4, it has many defects, some of which are rarely recognized. The most obvious problem lies in the heuristics used in pattern matching. These were originally introduced to avoid "futile" attempts to match (Griswold, 1972a, pp. 35–36, 126–131). The need for heuristics can be justified in theory by the observation that there are situations in which backtracking can cause pat-

tern matching to take arbitrarily long before failing, while the failure can nonetheless be predicted, a priori, on the basis of such a simple consideration as the length of the subject. Until SNOBOL4, these heuristics were strictly an implementation matter, since they did not affect the results of program execution except possibly in the amount of time required. With the introduction of unevaluated expressions and other dynamic effects during pattern matching in SNOBOL4, heuristics became a linguistic matter. Failure to complete pattern matching because of the heuristics might cause some component with significant effects to be bypassed. This problem was "solved" by placing the heuristics under the control of a programmable switch. The heuristics remained a problem, however. They are complex and difficult to understand (even the implementors have to resort to implementation listings to resolve some of the more subtle effects). The usefulness of the heuristics is also questionable. Experiments with a representative set of programs has shown that most programs run properly with or without the heuristics and the saving in execution time is small or nonexistent. It is quite possible that the overhead of supporting the heuristics is greater than the savings that they produce. The concern over the possible consequences of backtracking appears to be unjustified in practice, and the heuristics probably should have been removed.

Less obvious problems with pattern matching lie in its overly large vocabulary, its idiosyncrasies, and its lack of generality. The strict left-to-right direction of pattern matching and the lack of a facility for moving or resetting the focus of attention within the subject without advancing to the right except as a result of successful pattern matching produces awkward and inefficient programming techniques. The absence of synthesis facilities at the level of the analysis facilities results in an asymmetry and requires string construction to be carried out at an entirely different level and frame of reference from analysis. Doyle (1975) proposed solutions to some of these problems.

In addition, although analysis produces a complete "parse" of the subject according to the pattern, the results are unavailable to the programmer except in terms of success or failure and in the identification of specifically matched substrings. Thus the structure of the parse is lost. The power of pattern matching in SNOBOL4 is therefore somewhat illusory. It is trivial to write a recognizer for any context-free grammar (Griswold 1975a, pp. 7–12, 233–234), but the corresponding parser is much more difficult and requires use of side effects and unobvious techniques (Gimpel, 1976, pp. 411–415).

Another deficiency in pattern matching is the lack of any way to write source-language procedures to perform pattern matching. While defined functions can be used to extend the repertoire of built-in functions, there is no corresponding mechanism for extending the repertoire of built-in pattern-matching procedures. An approach to solving these problems has been recently suggested (Griswold, 1975e, 1976c, e).

The fact that decomposition of strings through pattern matching operates by the side effects has always been a problem. By its nature, this mechanism is error prone and unstructured. On the other hand, string analysis, by analogy with grammatical characterizations of languages, leads naturally to deriving multiple values from a single analysis. A well-structured, uncomplicated solution to this problem remains elusive.

### *3.3.4. Input and Output*

Input and output have always been problems in SNOBOL. Although the language constructs that perform input and output are particularly simple, there are two distinct problems: the formatting capabilities are weak, and there is no concept of files other than se-

quential ones. These deficiencies have restricted the use of SNOBOL4 in some cases and led to extensions in dialects such as SPITBOL.

The use of FORTRAN I/O as a specification for SNOBOL4 I/O had historical bases and was a mistake. It was chosen because FORTRAN was well known and so that implementors could use FORTRAN I/O packages instead of having to implement their own, thus enhancing portability. With advances in the state of the art, both of these concerns seem to have been misplaced. In retrospect it would have been a better choice to develop I/O specifically suitable for SNOBOL. Implementations that have departed from the FORTRAN standards have produced much more satisfactory results (Dewar, 1971; Gimpel, 1973c).

### 3.4. Changes and Additions

Many changes and additions to SNOBOL have been suggested, but not adopted. Most of these have been of the "junky" variety (such as a keyword &TEBAHPLA that would have contained an order-reversed copy of the alphabet). Many suggestions were made to meet specific or idiosyncratic needs. Our general attitude toward such suggestions was reserved, but they were accumulated with a mind toward abstracting the essence from apparently unrelated suggestions and producing unified improvements (Griswold, 1972b,c). For example, many suggestions were received for making specific improvements to pattern matching in SNOBOL3, but none of these was incorporated into the language. Nonetheless, a more general concept of patterns as data objects evolved. This concept subsumed all of the pattern-matching facilities of SNOBOL3, as well as the suggestions for changes, but it placed them in a conceptually simpler framework.

In summary, the general attitude was not to include specific language features, however useful they might appear, unless they fit naturally into the existing semantic framework or until a broader conceptual basis could be found. Phrased another way, the decision to include suggested features was more a matter of philosophy than pragmatism.

### 3.5. User Problems

Inefficiency is the problem most frequently ascribed to the SNOBOL languages, and interpretive implementations are frequently cited as the source of the problem. In fact, it is mainly language features that require run-time binding that lie at the root of most of the problems. Even "compiler" implementations such as SPITBOL (Dewar, 1971) have a significant interpretive component (Griswold, 1977a), and there are interpreters that rival the performance of such compilers (Gimpel, 1973b; Dewar and McCann, 1977).

Until SNOBOL4 there was, in fact, relatively little user concern about the efficiency of the SNOBOL languages. Despite the fact that third-generation hardware and subsequent developments have lowered the actual cost of computation considerably, increased awareness of costs and increased competition for resources resulted in more and more concern about efficiency. Thus SNOBOL4 is criticized for its inefficiency, even though it is actually substantially more efficient than SNOBOL3.

Nonetheless, the major problem that most users have had with SNOBOL4 *is* the resources that it requires. Depending on the user's environment, this may translate into cost, poor turnaround time, or the inability to run a program altogether due to storage

limitations. This problem extends to the unavailability of SNOBOL4 implementations on computers with limited memories. Naïve users, in particular, tend to get into difficulty with programs that run satisfactorily on small amounts of sample data, but are totally impractical when applied to real data. Although the developers of SNOBOL4 actively discouraged its use for large problems, recommending it instead for simple, "one-shot" uses (Poage, 1977), even when the advice was received it was seldom taken. An inherent problem is often overlooked: string data simply take large amounts of space. Programs are written, for example, to keep a dictionary in core and tests are made with only a few words. The user fails to see the consequences of using the full dictionary. This problem is, of course, not specific to SNOBOL4, but it is more severe there because of the emphasis on nonnumeric data and the ease of expressing otherwise complex operations on such data.

As mentioned in Section 1.9, the SIL implementation of SNOBOL4 was not expected to be efficient. In practice, this implementation carried SNOBOL4 very close to the fine line between practicality and impracticality. In a sense, the implementation succeeded almost too well. It is efficient enough for use in some environments and for some problems. As a consequence SNOBOL4 has come into wide use. On the other hand, it is too inefficient to run production programs in most environments. If the implementation had proved too inefficient for general use, early modification to the implementation or to the language itself might have produced a significantly more efficient system. To some extent, these concerns are of more historical than current interest, since more efficient implementations, notably SPITBOL, SITBOL, and MACRO SPITBOL are coming into widespread use (Griswold, 1976d).

### 3.6. Implementation Problems

Because of the design philosophy, which sometimes has been expressed as "the implementor be damned," SNOBOL has presented both serious and interesting implementation problems. Briefly summarized, these are:

1. Storage management, including varying length strings, dynamic allocation, and implicit garbage collection.
2. Maintenance of a changing symbol table at run time.
3. Implicit processes, such as input, output, and tracing.
4. Pattern matching.
5. Run-time compilation.
6. Delayed bindings, including redefinition of functions and operators at run time.

In addition, some language features have a disproportionate effect on implementation. For example, the ability to redefine any built-in operator at run time forces implementation techniques that impose a substantial execution overhead. The unevaluated expression operator provides a particularly glaring example. Since this operator must leave its operand unevaluated and any operation can be made synonymous with it at run time, in the SIL implementation all operations are handled in prefix mode. By simply exempting this operator from redefinition, and treating it specially during compilation, SITBOL allows more efficient suffix evaluation to be used in place of prefix evaluation (Griswold, 1977a). This exemption is certainly a minor concession of design to implementation effi-

ciency. Interestingly, however, prefix evaluation in SNOBOL4 preceded the introduction of unevaluated expressions and, in fact, it was the existence of prefix code that suggested unevaluated expressions.

Related issues and their treatment constitute a large and complex subject, which is discussed elsewhere in detail (Griswold, 1972a, 1977a).

### 3.7. Tradeoffs and Compromises

Very few of the traditional tradeoffs and compromises in language design were made for SNOBOL. By the nature of the design philosophy, few conscious linguistic concessions were made to implementation difficulties or efficiency. Most concessions, in fact, were made because of limited personnel resources and because of pressure (both from within and without the development group) to produce systems that could be used. Certainly several interesting areas for language extension went unexplored for lack of time and resources.

## 4. Implications for Current and Future Languages

### 4.1. Influences on Other Languages

The SNOBOL languages have had little apparent direct effect on the mainstream of programming language development. This lack of direct influence is probably a necessary consequence of the direction SNOBOL has taken, which is deliberately contrary to the mainstream. Most of the direct influence has been reflected in offshoots and dialects of existing languages.

String processing extensions to FORTRAN represent the most prevalent influence of the basic string operations and pattern matching (Jessup, 1966; Puckett and Farlow, 1966; Zweig, 1970; Gimpel, 1970; Shapiro, 1970; Baron and Golini, 1970; Hanson, 1974). Similar proposals have been made for PL/I (Rosin, 1967), ALGOL 60 (Storm, 1968), and ALGOL 68 (Dewar, 1975a).

The obvious relationship between SNOBOL and macro processing has resulted in the use of SNOBOL-style pattern matching in a number of macro processors (Waite, 1967, 1969; Kagan, 1972; Brown, 1974; Wilson, 1975).

Where there has been official acceptance of SNOBOL-like features in major languages, it has been grudging. PL/I is an example. The designs of SNOBOL4 and PL/I were contemporaneous, although the mechanics of the designs were very different. While SNOBOL, SNOBOL2, and SNOBOL3 had for several years supported true varying length strings, the designers of PL/I were reluctant to accept this concept presumably because of implementation considerations. I particularly recall Dave Farber and Doug McIlroy urging that string length be handled implicitly in PL/I but to no avail (although the EPL dialect of PL/I (Cambridge Information Systems Laboratory, 1968) did dynamically allocate varying strings). Similarly, Bob Rosin carried on an early campaign for simple string matching functions (Rosin, 1967), but only recently has something of this kind been added in the form of AFTER and BEFORE (American National Standards Institute, 1976, pp. 318, 322). COBOL also has expanded its earlier EXAMINE facility to a more powerful INSPECT statement, along with STRING and UNSTRING statements for decomposition

and concatenation (American National Standards Institute, 1974, Ch. II, pp. 68–73, 86–88, 91–94).

The influence of SNOBOL features on dialects and divergent language developments has been more marked, if less influential. The SNOBOL concept of patterns has been most significant. Notable examples occur in PL/I (Pfeffer and Furtado, 1973) and LISP (Smith and Enea, 1973; Tesler *et al.*, 1973).

Incorporation of SNOBOL features or philosophy into other language contexts has led to more radical departures for the conventional programming languages. APLBOL, for example, provides an amalgamation of APL and SNOBOL (Leichter, 1976). ESP3 extends the concept of SNOBOL4 pattern matching to three-dimensional objects (Shapiro, 1976; Shapiro and Baron, 1977).

Although SNOBOL has never enjoyed general popularity in the AI community, it has had an indirect, although evident, influence on AI languages such as 1.pak and 2.pak (Mylopoulos *et al.*, 1973; Melli, 1974). The usefulness of a simple SNOBOL-like language for AI work is evident in EASEy (Uhr, 1974).

Even the mainstream of SNOBOL development is not entirely dead. Development of a better understanding of pattern matching (Druseikis and Doyle, 1974; Griswold, 1975e) has led to the development of SL5 (*S*NOBOL *L*anguage *5*) in the spirit of SNOBOL but specifically designed for research rather than for general use (Griswold *et al.*, 1977; Griswold, 1976c,e).

An important influence of SNOBOL has been its "orthogonality" to the mainstream of programming language development. Where there has been a movement for strong typing, SNOBOL has offered a "typeless" alternative. Where concern over program structure and verification has led to the condemnation of a number of "harmful" features and encouraged a "police-state" school of programming, SNOBOL has provided ease of programming and freedom from constraints. This is not to say that the current concern with structured programming is misplaced, but rather that there are a variety of concerns that make different alternatives appropriate in different situations.

SNOBOL has shown that language features, once thought to be impractical because of implementation considerations, are in fact both practical and valuable. Specifically, SNOBOL has led to the development of implementation techniques to meet challenges of language features. In short, as a "maverick," SNOBOL has offered an alternative and has led many persons to think of possibilities that might not have been considered otherwise.

In the area of pattern matching, SNOBOL has directed attention toward backtrack control structures (most notably in AI languages) and to the implementation of such features (Druseikis and Doyle, 1974). The resulting interest in the use of coroutine mechanisms for implementing search and backtracking algorithms (Griswold, 1976f; Hanson, 1976b; Hanson and Griswold, 1978) may have substantial future impact on programming language design.

Of all the features of SNOBOL4, one of the most important in practice has proved to be the table data type, which provides at the source level a feature similar to the internal symbol tables of assemblers and compilers. By eliminating the need for the programmer to handle all the tedious and error-prone details of symbol tables at the source-language level, SNOBOL4 has provided a way in which symbol table techniques can be used freely and in which concise, simple programs can be written where the programming burden would otherwise be unsupportable (Griswold, 1975a, pp. 42–43, 194–197). Probably no other single feature of SNOBOL4 has made programming simpler or opened the door to

more application areas. I consider tables to be a prime candidate for inclusion in new high-level languages.

### 4.2. Influence on Implementation Techniques

The problems inherent in the efficient implementation of SNOBOL-like languages have stimulated the development of new techniques and have had the indirect effect of liberating the design of programming languages from many of the constraints formerly attributable to problems of implementation and efficiency. For example, automatic storage management was considered to be a major implementation problem at the time SNOBOL was designed. Now, due at least in part to SNOBOL, the concern is no longer a major issue (Griswold, 1972a, Hanson, 1977a).

Efficient interpretive systems and particularly the handling of run-time bindings have been considerably developed as a result of SNOBOL4 implementations (Gimpel, 1973b; Dewar, 1975b; Dewar and McCann, 1977). A number of other interesting implementation techniques have been developed (Griswold, 1972a, 1977a; Tye, 1972; Gimpel, 1973b, 1974b; Druseikis and Doyle, 1974; Sears, 1974, Hanson, 1976a; Silverston, 1976b).

### 4.3. Influences on Theoretical Work

While SNOBOL by its very nature is beyond the realm of the verificationists, it has posed some interesting problems in the description of formal semantics (Herriot, 1973a,b, 1974; Mickel, 1973; Pagan, 1978; Tennent, 1973). Most theoretical work has concentrated on the description of patterns and pattern matching (Gimpel, 1973a, 1975; Goyer, 1973; Tennent, 1973; Stewart, 1975; Fleck, 1977), their implementation (Goyer, 1973; Druseikis and Doyle, 1974), and on the relationships between patterns and data structures (Fleck, 1971; Fleck and Liu, 1973).

### 4.4. Advice from the Experience of SNOBOL

Other language designers should look to the success and durability of a language that is as contrary as SNOBOL for possible future directions. In my opinion, programming language development has a long way to go before it achieves "perfection"—if that is meaningful, possible, desirable, or even definable. There is too much of a tendency to get caught up with the fad of the moment, whether it is goto-less programming, FORTRAN preprocessors for structured programming, or data abstraction. Someone needs to look at alternatives, to consider unpopular views, and perhaps, particularly, to seek unconventional approaches. SNOBOL serves as an example of the success of such an endeavor.

### 4.5. The Long-Range Future of SNOBOL

The long-range future of SNOBOL depends on several issues. For example, SNOBOL, like FORTRAN, might prove indestructible simply because of the extent of its acceptance, regardless of the availability of better languages. I regard that as possible but unlikely. It is likely that a new language will be developed that both shares the philosophy of SNOBOL and is also better at doing the things SNOBOL was designed to do. After all, SNOBOL3 replaced SNOBOL and SNOBOL4 has virtually replaced SNOBOL3.

It is also possible that the current interest (or fad) for structured programming will eventually erode the basis of support of SNOBOL so that it will gradually fall into disuse. There is little evidence that this is happening at the present and I regard this future possibility as unlikely.

Here it is worthwhile to consider again the major reason for the success of SNOBOL—its availability. There have been many "good" programming languages that have been designed but not implemented. Others have been implemented but not made generally available. Furthermore there are generally available languages that are not maintained or supported. SNOBOL is in the small class of nonvendor-supported languages that are available on a wide variety of machines and that are well documented and consistently supported (Griswold, 1978a). It is my personal opinion that it was the general availability of SNOBOL processors and language documentation that were responsible, more than any other factor, for SNOBOL's initial success. With the transition to SNOBOL4, the wide range of readily available SIL implementations further extended SNOBOL's use. Efficient implementations of SNOBOL4, most notably SPITBOL, significantly enlarged its audience. The increasing availability of smaller computers now raises the question of the availability of SNOBOL processors to the growing ranks of programmers. The recent development of implementations of SNOBOL4 for the PDP-11 (Dewar, 1977; Dalgleish, 1977) and work on implementations for other mid-range machines may be an important determining factor in the durability of SNOBOL4.

## ACKNOWLEDGMENTS

The material generously supplied to me by a number of persons over a period of years has proved immensely valuable in the preparation of the paper. The questions, comments, suggestions, and critical reviews of this paper by Dave Farber, Dave Hanson, Doug McIlroy, Jim Poage, Ivan Polonsky, Bob Rosin, Jean Sammet, Mike Shapiro, and Hank Tropp have been invaluable. I am, of course, solely responsible for errors of omission and commission. I am indebted most of all to my wife, Madge. She has helped with all aspects of the development of the paper: as a historian, editor, critic, organizer, and keyboarder. Most of all she has provided constant support and encouragement during the entire project.

## REFERENCES

Abrahams, P. W. (1974). Improving the control structure of SNOBOL4. *SIGPLAN NOTICES* 9(5): 10–12.

American National Standards Institute (1974). *American National Standard Programming Language COBOL*, ANSI X3.23-1974, New York.

American National Standards Institute (1976). *American National Standard Programming Language PL/I*, ANSI X3.53-1976, New York.

Anderson, R. O., and Griswold, R. E. (1976) February 18. *ACOLYTE; A Document Formatting Program.* Tucson, Arizona: University of Arizona, Department of Computer Science. SNOBOL4 Project Doc. S4PD11b.

Bailey, F. N., Brann, J., and Kain, R. Y. (1969) August 10. *Algebra I Users Reference Manual.* Minneapolis, Minnesota: University of Minnesota. Department of Electrical Engineering.

Barber, C. L. R. (1973) December 10. *SNOBOL4 Version 3.7.* El Segundo, California: XDS User's Group. Program Library Catalog Number 890823-11A00.

Baron, R. J., and Golini, J. (1970) September. *Strings: Some FORTRAN Callable String Processing Routines* (Unpublished technical report). Iowa City, Iowa: University of Iowa, Department of Computer Science.

Berndl, W. (1975) October. *An Analysis of the SPITBOL System.* Toronto, Ontario: Department of Computer Science, University of Toronto. Technical Report No. 85.

Beyer, T. (1976) April 30. *SNECS.* Letter to R. E. Griswold.

Brown, P. J. (1974) November. *Towards More General String Manipulation—SNOBOL4 Compared With ML/I* (Unpublished technical report). Canterbury, England: University of Kent at Canterbury.

Calise, M. F. (1966) February 11. *Disk Functions for SNOBOL3*. (Unpublished internal memorandum). Holmdel, New Jersey: Bell Laboratories.

Cambridge Information Systems Laboratory (1968) April. *ELP Users' Reference Manual*. Cambridge, Massachusetts.

Chai, D., and DiGiuseppe, J. (1965) May. *A Study of System Design Considerations in Computers for Symbol Manipulation*. Ann Arbor, Michigan: University of Michigan, Department of Electrical Engineering. Report No. 05635-1-F.

Conrow, K. (1977). A FAMULUS post-processor. *SIGDOC Newsletter* **4**(3): 7–8.

Corbató, F. J., and Vyssotsky, V. A. (1965). Introduction and overview of the MULTICS system. In *AFIPS Conference Proceedings, Fall Joint Computer Conference*, pp. 185–196. Washington, D.C.: Spartan Books.

Corbató, F. J., *et al.* (1963). *The Compatible Time-Sharing System: A Programmer's Guide*. Cambridge, Massachusetts: MIT Press.

Croff, D. L. (1974) November. *SNOFLEX Handbook* (unpublished technical report). Eugene, Oregon: University of Oregon, Department of Computer Science.

Dalgleish, R. (1977) September 15. Letter to R. E. Griswold.

Datacraft Corporation (1972) July. *Series 6000 SNOBOL4 General Specification*. Fort Lauderdale, Florida.

Dewar, C. E. S. (1977). *RE: Release of SPITBOL-11* Chicago, Illinois: Dewar Information Systems Corporation.

Dewar, R. B. K. (1971) February 12. *SPITBOL Version 2.0* (SNOBOL4 Project Document S4D23). Chicago, Illinois: Illinois Institute of Technology.

Dewar, R. B. K. (1975a). *String Processing in ALGOL-68* (Unpublished technical report). Chicago, Illinois: Illinois Institute of Technology.

Dewar, R. B. K. (1975b). Indirect threaded code. *Communications of the ACM* **18**(6): 330–331.

Dewar, R. B. K., and McCann, A. P. (1974) December. *1900 SPITBOL*. Leeds, England: University of Leeds, Centre for Computer Studies. Technical Report No. 55.

Dewar, R. B. K., and McCann, A. P. (1977). Macro SPITBOL—a SNOBOL4 compiler. *Software—Practice and Experience* **7**: 95–113.

Dewar, R. B. K., Belcher, K., and Cole, J. (1975) March. *UNIVAC/SPITBOL;* Version 1.0. Chicago, Illinois: Illinois Institute of Technology, Department of Computer Science.

Dickman, B. N., and Jensen, P. D. (1968) January 9. *Tracing Facilities for SNOBOL4* (Unpublished Technical Memorandum 68-3344-1). Holmdel, New Jersey: Bell Laboratories.

Doyle, J. N. (1975) February 11. *A Generalized Facility for the Analysis and Synthesis of Strings, and a Procedure Based Model of an Implementation*. Tucson, Arizona: University of Arizona, Department of Computer Science. SNOBOL4 Project Document S4D48.

Druseikis, F. C., and Doyle, J. N. (1974). A procedural approach to pattern matching in SNOBOL4. In *Proceedings of the ACM Annual Conference*, pp. 311–317. New York: Association for Computing Machinery.

Druseikis, F. C., and Griswold, R. E. (1973) October 11. *An Extended Function Definition Facility for SNOBOL4*. Tucson, Arizona: University of Arizona, Department of Computer Science. SNOBOL4 Project Document S4D36.

Druseikis, F. C., and Griswold, R. E. (1977) August 24. *SPITBOL 6000 User's Manual* (Unpublished technical report). Tucson, Arizona: University of Arizona, Department of Computer Science.

Dunn, R. (1973). SNOBOL4 as a language for bootstrapping a compiler. *SIGPLAN Notices* **8**(5): 28–32.

Farber, D. J. (1963a) October 8. *FORTRAN Compiler for "Double Presision* (sic) *Project"* (Program listing). Holmdel, New Jersey: Bell Laboratories.

Farber, D. J. (1963b) October 18. SNOBOL, an improved COMIT-like language (oral presentation). Ann Arbor, Michigan: University of Michigan, Computer Center.

Farber, D. J., Griswold, R. E., and Polonsky, I. P. (1963a) May 16. *A Preliminary Report on the String Manipulation Language SNOBOL* (Unpublished Technical Memorandum 63-3344-2). Holmdel, New Jersey: Bell Laboratories.

Farber, D. J., Griswold, R. E., and Polonsky, I. P. (1963b) October 24. Letter to R. W. Hamming.

Farber, D. J., Griswold, R. E., and Polonsky, I. P. (1964a). SNOBOL, a string manipulation language. *Journal of the ACM* **11**(1): 21–30.

Farber, D. J., Griswold, R. E., and Polonsky, I. P. (1964b) April. *SNOBOL 2* (sic) (Unpublished internal memorandum). Holmdel, New Jersey: Bell Laboratories.

Farber, D. J., Griswold, R. E., and Polonsky, I. P. (1964c) April 28. Internal memorandum. Holmdel, New Jersey: Bell Laboratories.

Farber, D. J., Griswold, R. E., and Polonsky, I. P. (1964d) October 13. *SNOBOL3* (Unpublished Technical Memorandum 64-3344-1). Holmdel, New Jersey: Bell Laboratories.

Farber, D. J., Griswold, R. E., and Polonsky, I. P. (1965a) October 7. *SNOBOL3 Source* (Program listing). Holmdel, New Jersey: Bell Laboratories.

Farber, D. J., *et al.*, (1965b) May 13. *Programming Machine-Language Functions for SNOBOL3* (Unpublished Technical Memorandum 64-3343-2). Holmdel, New Jersey: Bell Laboratories.

Farber, D. J., Griswold, R. E., and Polonsky, I. P. (1966). SNOBOL3 programming language. *Bell System Technical Journal* **45**: 895–944.

Faulhaber, G. R. (1963) August 26. *SNAFU* (program listing). Holmdel, New Jersey: Bell Laboratories.

Faulhaber, G. R. (1964) February 17. *A Simulation Program for One- and Two-Dimensional Automata* (Unpublished Engineers Notes). Holmdel, New Jersey: Bell Laboratories.

Flannery, M. G. (1964) July 22. *Push and Pop* (Unpublished Engineers Notes). Holmdel, New Jersey: Bell Laboratories.

Fleck, A. C. (1971). Towards a theory of data structures. *Journal of Computer and System Sciences* **5**: 475–488.

Fleck, A. C. (1977) March. *Formal Models of String Patterns* (Unpublished technical report). Iowa City, Iowa: University of Iowa, Computer Science Department and University Computer Center.

Fleck, A. C., and Liu, L.-C. (1973) June. *On the Realization of Data Graphs*. Iowa City, Iowa: University of Iowa, Department of Mathematics. Technical Report No. 67.

Forte, A. (1967). *SNOBOL3 Primer; An Introduction to the Computer Programming Language*. Cambridge, Massachusetts: MIT Press.

Gaines, R. S. (1968) March 1. *Preliminary Report on the SNOBOL4 Programming Language, Revised to Conform to the CDC 6000 Implementation* (Unpublished technical report). Princeton, New Jersey: Institute for Defense Analyses.

Galler, B. A. (1968) March. Letter to the Editor. *Communications of the ACM* **11**(3): 148

Galler, B. A., and Perlis, A. J. (1970). *A View of Programming Languages*. Reading, Massachusetts: Addison-Wesley.

Gaskins. R., Jr. (1970) March. *CAL SNOBOL Reference Manual* (Unpublished technical report). Berkeley, California: University of California, Computer Center.

Gimpel, J. F. (1970) March 23. SNOBOLizing FORTRAN. Letter to R. Zweig.

Gimpel, J. F. (1972). Blocks—A new datatype for SNOBOL4. *Communications of the ACM* **15**(6): 438–447.

Gimpel, J. F. (1973a). A theory of discrete patterns and their implementation in SNOBOL4. *Communications of the ACM* **16**(2): 91–100.

Gimpel, J. F. (1973b) May 10. *A Design for SNOBOL4 for the PDP-10, Part I—The General.* Holmdel, New Jersey: Bell Laboratories. SNOBOL4 Project Document S4D29b.

Gimpel, J. F. (1972c) June 1. *SITBOL Version 3.0.* Holmdel, New Jersey: Bell Laboratories. SNOBOL4 Project Document S4D30b.

Gimpel, J. F. (1973d). Private communication to R. E. Griswold.

Gimpel, J. F. (1974a). The minimization of spatially-multiplexed character sets. *Communications of the ACM* **17**(6): 315–318.

Gimpel, J. F. (1974b) May 1. *A Hierarchical Approach to the Design of Linkage Conventions.* Holmdel, New Jersey: Bell Laboratories. SNOBOL4 Project Document S4D41.

Gimpel, J. F. (1974c). Some highlights of the SITBOL language extensions for SNOBOL4. *SIGPLAN Notices* **9**(10): 11–20.

Gimpel, J. F. (1975). Nonlinear pattern theory. *Acta Informatica* **4**: 213–229.

Gimpel, J. F. (1976). *Algorithms in SNOBOL4*. New York: Wiley.

Gimpel, J. F., and Hanson, D. R. (1973) October 5. *The Design of ELFBOL—A Full SNOBOL4 for the PDP-11.* Holmdel, New Jersey: Bell Laboratories. SNOBOL4 Project Document S4D43.

Goyer, P. (1973) August. *Le language SNOBOL4 et la conformité chaîne-modèle*. Ph. D. Thesis, Université de Montreal.

Griswold, R. E. (1962) September 18. *The Separation of Flow Graphs.* (Unpublished Technical Memorandum 62-3344-3). Holmdel, New Jersey: Bell Laboratories.

Griswold, R. E. (1963a) March. *SCL7* (Unpublished draft). Holmdel, New Jersey: Bell Laboratories.

Griswold, R. E. (1963b) April. *A Preliminary Report on a String Manipulation Language* (Unpublished draft). Holmdel, New Jersey: Bell Laboratories.

Griswold, R. E. (1963c) July 18. *Algebraic Evaluation* (Program listing). Holmdel, New Jersey: Bell Laboratories.

Griswold, R. E. (1964a) January 24. *Syntax Analysis* (Program listing). Holmdel, New Jersey: Bell Laboratories.
Griswold, R. E. (1964b) August 4. *Determinant Computation Using 7/29/63 SNOBOL3* (Program listing). Holmdel, New Jersey: Bell Laboratories.
Griswold, R. E. (1964c) October 6. *SNOBOL2 is Obsolete* (Program listing). Holmdel, New Jersey: Bell Laboratories.
Griswold, R. E. (1965) June 1. *Linked-List Functions for SNOBOL3* (Unpublished Technical Memorandum 65-3343-6). Holmdel, New Jersey: Bell Laboratories.
Griswold, R. E. (1966–1969). Entries in *SNOBOL4 Project Log*. Holmdel, New Jersey: Bell Laboratories.
Griswold, R. E. (1966a) April 18. *Special Purpose SNOBOL3 Functions*—II (Unpublished Technical Memorandum 65-3343-1). Holmdel, New Jersey: Bell Laboratories.
Griswold, R. E. (1966b) April 24. Entry in *SNOBOL4 Project Log*. Holmdel, New Jersey: Bell Laboratories.
Griswold, R. E. (1966c) May 9. *Tentative SNOBOL4 Syntax Described in SNOBOL4* (Unpublished draft). Holmdel, New Jersey: Bell Laboratories.
Griswold, R. E. (1966d) July 28. Entry in *SNOBOL4 Project Log*. Holmdel, New Jersey: Bell Laboratories.
Griswold, R. E. (1967) June 28. Entry in *SNOBOL4 Distribution Log*. Holmdel, New Jersey: Bell Laboratories.
Griswold, R. E. (1968–). *SNOBOL4 Information Bulletins*. Holmdel, New Jersey: Bell Laboratories and Tucson, Arizona: University of Arizona, Department of Computer Science. (Published irregularly.)
Griswold, R. E. (1968) January 1. Entry in *SNOBOL4 Distribution Log*. Holmdel, New Jersey: Bell Laboratories.
Griswold, R. E. (1970) February 27. *A Guide to the Macro Implementation of SNOBOL4* (Unpublished Technical Memorandum 70-8242-5). Holmdel, New Jersey: Bell Laboratories.
Griswold, R. E. (1971). *MAIN 79* (Program listing). Holmdel, New Jersey: Bell Laboratories.
Griswold, R. E. (1972a). *The Macro Implementation of SNOBOL4: A Case Study of Machine-Independent Software Development*. San Francisco, California: Freeman.
Griswold, R. E. (1972b) November 9. *Suggestions for New Features in SNOBOL4* (Unpublished technical report). Tucson, Arizona: University of Arizona, Department of Computer Science.
Griswold, R. E. (1972c) November 13. *Suggestions for New Features in SNOBOL4; Round 2—The Embellisher's Delight* (Unpublished technical report). Tucson, Arizona: University of Arizona, Department of Computer Science.
Griswold, R. E. (1974). Suggested revisions and additions to the syntax and control mechanism of SNOBOL4. *SIGPLAN Notices* **9**(2): 7–23.
Griswold, R. E. (1975a). *String and List Processing in SNOBOL4: Techniques and Applications*. Englewood Cliffs, New Jersey: Prentice-Hall.
Griswold, R. E. (1975b). A portable diagnostic facility for SNOBOL4. *Software—Practice and Experience* **5**: 93–104.
Griswold, R. E. (1975c) February 5. *Additions to SNOBOL4 to Facilitate Programming Language Research*. Tucson, Arizona: University of Arizona, Department of Computer Science. SNOBOL4 Project Document S4D37c.
Griswold, R. E. (1975d) May 22. *GENLAB II; A Program for Synthesizing Text*. Tucson, Arizona: University of Arizona, Department of Computer Science. SNOBOL4 Project Document S4PD9a.
Griswold, R. E. (1975e). Extensible pattern matching in SNOBOL4. In *Proceedings of the ACM Annual Conference*, pp. 248–252. New York: Association for Computing Machinery.
Griswold, R. E. (1976a). The macro implementation of SNOBOL4. In *Software Portability*, pp. 180–191. Cambridge, England: Cambridge University Press.
Griswold, R. E. (1976b) June 9. *Source and Cross-Reference Listings for the SIL Implementation of SNOBOL4*. Tucson, Arizona: University of Arizona, Department of Computer Science. SNOBOL4 Project Document S4D26b.
Griswold, R. E. (1976c) June 17. *String Scanning in SL5*. Tucson, Arizona: University of Arizona, Department of Computer Science. SL5 Project Document S5LD5a.
Griswold, R. E. (1976d) July 28. *SNOBOL4 Information Bulletin S4B17*. Tucson, Arizona: University of Arizona, Department of Computer Science.
Griswold, R. E. (1976e). String analysis and synthesis in SL5. In *Proceedings of the ACM Annual Conference*, pp. 410–414. New York: Association for Computing Machinery.
Griswold, R. E. (1976f). The SL5 programming language and its use for goal-oriented programming. In *Proceedings of the Fifth Texas Conference on Computing Systems*, pp. 1–5. Austin, Texas: University of Texas.

Griswold, R. E. (1977a) February 4. *Highlights of Two Implementations of SNOBOL4.* Tucson, Arizona: University of Arizona, Department of Computer Science. SNOBOL4 Project Document S4D55.

Griswold, R. E. (1977b) December 9. *Bibliography of Documents Related to the SNOBOL Programming Languages* (Unpublished draft). Tucson, Arizona: University of Arizona, Department of Computer Science.

Griswold, R. E. (1978a) January 7. *Sources of Implementations of SNOBOL4.* Tucson, Arizona: University of Arizona, Department of Computer Science. SNOBOL4 Project Document S4N21f.

Griswold, R. E. (1978b) January 11. *Bibliography of Numbered SNOBOL4 Documents; May 1967 through January 1978.* Tucson, Arizona: University of Arizona, Department of Computer Science. SNOBOL4 Project Document S4D43b.

Griswold, R. E., and Griswold, M. T. (1973). *A SNOBOL4 Primer.* Englewood Cliffs, New Jersey: Prentice-Hall.

Griswold, R. E., and Hanson, D. R. (1977). An Overview of SL5. *SIGPLAN Notices* **12**(4): 40–50.

Griswold, R. E., and Polonsky, I. P. (1962) September 5. *The Classification of the States of a Finite Markov Chain* (Unpublished Technical Memorandum 62-3344-3). Holmdel, New Jersey: Bell Laboratories.

Griswold, R. E., and Polonsky, I. P. (1963a) July 1. *String Pattern Matching in the Programming Language SNOBOL* (Unpublished Technical Memorandum 63-3344-3). Holmdel, New Jersey: Bell Laboratories.

Griswold, R. E., and Polonsky, I. P. (1963b) September 24. *IPL Data Compiler* (Program listing). Holmdel, New Jersey: Bell Laboratories.

Griswold, R. E., and Polonsky, I. P. (1965) February 1. *Tree Functions for SNOBOL3* (Unpublished Technical Memorandum 65-3343-1). Holmdel, New Jersey: Bell Laboratories.

Griswold, R. E., and Varian, L. C. (1964) November 24. *Special Purpose SNOBOL3 Functions* (Unpublished Technical Memorandum 64-3344-2). Holmdel, New Jersey: Bell Laboratories.

Griswold, R. E., Poage, J. F., and Polonsky, I. P. (1967a) May 1. *Preliminary Description of the SNOBOL4 Programming Language* (Unpublished Technical Memorandum 67-3344-2). Holmdel, New Jersey: Bell Laboratories.

Griswold, R. E., Poage, J. F., and Polonsky, I. P. (1967b) May 1. *Preliminary Description of the SNOBOL4 Programming Language.* Holmdel, New Jersey: Bell Laboratories. SNOBOL4 Project Document S4D1.

Griswold, R. E., Poage, J. F., and Polonsky, I. P. (1967c) October 20. *Preliminary Report on the SNOBOL4 Programming Language—II.* Holmdel, New Jersey: Bell Laboratories. SNOBOL4 Project Document S4D4.

Griswold, R. E., Poage, J. F., and Polonsky, I. P. (1968a) August 8. *The SNOBOL4 Programming Language.* Holmdel, New Jersey: Bell Laboratories. SNOBOL4 Project Document S4D9.

Griswold, R. E., Poage, J. F., and Polonsky, I. P. (1968b). *The SNOBOL4 Programming Language.* Englewood Cliffs, New Jersey: Prentice-Hall.

Griswold, R. E., Poage, J. F., and Polonsky, I. P. (1971). *The SNOBOL4 Programming Language,* Second Edition. Englewood Cliffs, New Jersey: Prentice-Hall.

Griswold, R. E., Hanson, D. R., and Korb, J. T. (1977) October 18. *An Overview of the SL5 Programming Language.* Tucson, Arizona: University of Arizona, Department of Computer Science. SL5 Project Document S5LD1d.

Guard, J. R. (1967) December 29. *SNOBOL.* Princeton, New Jersey: Applied Logic Corporation. Program Bulletin #67-006.

Haight, R. C. (1970) October 21. *The SNOFLAKE Programming Language* (Unpublished Technical Memorandum 70-9155-2). Piscataway, New Jersey: Bell Laboratories.

Hall, J. T., and McCabe, P. S. (1967) October 17. *SNOBOL3 Primitive Functions—Binary Routines Store on Disk* (Unpublished Technical Memorandum 67-5744-1). Indian Hill, Illinois: Bell Laboratories.

Hallyburton, J. C., Jr. (1974). *Advanced Data Structure Manipulation Techniques for the SNOBOL4 Programming Language.* Ph.D. Thesis, University of Arizona, Department of Computer Science.

Hamlin, K. B. (1965) February 16. Letter to W. Keister and M. E. Terry.

Hamming, R. W. (1963) October 25. Letter to D. J. Farber, R. E. Griswold, and I. P. Polonsky.

Hanson, D. R. (1973) June 8. Letter to the editor of *SIGPLAN Notices.*

Hanson, D. R. (1974). A simple technique for representing strings in FORTRAN IV. *Communications of the ACM* **17**(11): 646–647.

Hanson, D. R. (1976a). Variable associations in SNOBOL4. *Software—Practice and Experience* **6**: 245–254.

Hanson, D. R. (1976b). A procedure mechanism for backtrack programming. In *Proceedings of the ACM Annual Conference,* pp. 401–405. New York: Association for Computing Machinery.

Hanson, D. R. (1977a). Storage management for an implementation of SNOBOL4. *Software—Practice and Experience* **7**: 179–192.

Hanson, D. R. (1977b). RATSNO—an experiment in software adaptability. *Software—Practice and Experience* **7**: 623–630.

Hanson, D. R., and Griswold, R. E. (1978). The SL5 procedure mechanism. *Communications of the ACM,* **21**(5): 392–400.

Herriot, R. G. (1973a). Gloss: a semantic model of programming languages. *SIGPLAN Notices* **8**(9): 70–73.

Herriot, R. G. (1973b). Gloss: a high level machine. *SIGPLAN Notices* **8**(11): 81–90.

Herriot, R. G. (1974). A uniform view of control structures in programming languages. In *Proceedings of IFIP Congress, Stockholm, 74,* pp. 331–335.

Hirsch, A. E., Jr. (1973) May 15. Letter to G. L. Baldwin.

IBM Corporation (1969). *System/360 Administrative Terminal System—OS; Terminal Operations Manual.* White Plains, New York, Application Program H20-0589-1.

IBM Corporation (1970). *PL/I(F) Language Reference Manual.* White Plains, New York, File No. 5360-29.

Jessup, R. F. (1966) November 30. *SNIFF, A Set of Subroutines for String Operations in FORTRAN* (Unpublished Technical Memorandum 66-6322-9). Holmdel, New Jersey: Bell Laboratories.

Kagan, C. A. R. (1972). The multigap extension to string language processors. *SIGPLAN Notices* **3**(3): 115–146.

Kain, R. Y., and Bailey, F. N. (1967) September 12. *SNOBOL 67 Users Reference Manual* (Unpublished technical report). Minneapolis, Minnesota: University of Minnesota, Department of Electrical Engineering.

Keister, W. (1964) October 20. Letter to R. E. Griswold.

Keister, W. (1970). Private communication to R. E. Griswold.

Kernighan, B. W., and Plauger, P. J. (1976). *Software Tools.* Reading, Massachusetts: Addison-Wesley.

Kersey, G. (1967). Private communication to R. E. Griswold.

Kostalansky, E. (1967). The definition of the syntax and semantics of the language SNOBOL I (in Slovak). *Kybernetika* **3**.

Lampson, B. W. (1966) April 18. *930 SNOBOL System Reference Manual.* Berkeley, California: University of California. Document 30.50.70, Contract No. SD-185, ARPA.

Lavrov, S. S. (1968). *SNOBOL-A; A String Manipulation Language* (in Russian). Moscow: USSR Academy of Science Computer Center.

Lee, C. Y. (1960). Automata and finite automata. *Bell System Technical Journal* **39**: 1276–1296.

Lee, C. Y. (1961a). An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers* EC-**10**: 346–365.

Lee, C. Y. (1961b). Categorizing automata by W-machine programs. *Journal of the ACM* **10**(8): 384–399.

Lee, C. Y. (1963) April 14. Handwritten comments on *A Preliminary Report on a String Manipulation Language.* (See Griswold 1963b.)

Lee, C. Y., and Paull, M. C. (1963). A content addressable distributed logic memory with applications to information retrieval. *Proceedings of the IEEE* **51**(6): 924–932.

Lee, C. Y., *et al.* (1962) September 1. *A Language for Symbolic Communication* (Unpublished Technical Memorandum 62-3344-4). Holmdel, New Jersey: Bell Laboratories.

Leichter, J. (1976). *APLBOL* (Unpublished technical report). Waltham, Massachusetts: Brandeis University, Mathematics Department.

LeSeur, W. J. (1969) April 10. *Text 360.* White Plains, New York: IBM Corporation. Document 360D-29.4.001.

Lindsay, J. H. (1975). *SNOBOLY; A Counterproposal to SNOBOLX* (Unpublished technical report). Kingston, Ontario: Queen's University, Department of Computing and Information Science.

Madnick, S. E. (1966) June. *SPL/I: A String Processing Language.* Cambridge, Massachusetts: IBM Corporation, Cambridge Scientific Center. Report 36.006.

Magnani, R. (1964) March 2. *A SNOBOL Program for Detecting Isomorphism Between Pairs of Weighted, Directed Graphs* (Unpublished Technical Memorandum 64-3341-1). Holmdel, New Jersey: Bell Laboratories.

Manacher, G. K. (1963) October 14. *Syntactic Functions* (Program listing). Holmdel, New Jersey: Bell Laboratories.

Manacher, G. K. (1964) July 1. *A Package of Subroutines for the SNOBOL Language* (Unpublished Technical Memorandum 64-1222-4). Holmdel, New Jersey: Bell Laboratories.

Manacher, G. K., and Varian, L. C. (1964) October 23. *A Dimension Statement and Random Number Facility for the SNOBOL Language* (Unpublished Technical Memorandum 64-1222-10). Holmdel, New Jersey: Bell Laboratories.

Martellotto, N. A. (1966) June 9. *SNOBOL Questionnaire* (Internal memorandum). Holmdel, New Jersey: Bell Laboratories.

Maurer, W. D. (1976). *A Programmer's Introduction to SNOBOL.* New York: Am. Elsevier.

McCann, A. P., Holden, S. C., and Dewar, R. B. K. (1976) December. *MACRO SPITBOL—DECsystem-10 Version.* Leeds, England: University of Leeds, Centre for Computer Studies. Technical Report No. 94.

McIlroy, M. D. (1962) August 7. *A String Manipulation System for FAP Programs* (Unpublished Technical Memorandum 62-1271-4). Holmdel, New Jersey: Bell Laboratories.

McIlroy, M. D. (1963). A variant method for file searching. *Communications of the ACM* **6**(3): 101.

McIlroy, M. D. (1977) December 5. Letter to R. E. Griswold.

Melli, L. F. (1974) December. *The 2.pak Language Primitives for AI Applications,* Masters Thesis, University of Toronto, Department of Computer Science.

Mickel, A. B. (1973) August. *Comparative Study of the Semantics of Selected Programming Languages.* Minneapolis, Minnesota: University of Minnesota, Computer, Information and Control Sciences. Technical Report TR 73-9.

MIT Press (1962). *An Introduction to COMIT Programming.* Cambridge, Massachusetts.

Moody, J. K. M. (1972) November 1. *SNOBOL4 on Titan* (Unpublished technical report). Cambridge, England: University of Cambridge, Computer Laboratory.

Mooers, C. N. (1968) March. Reply to letter to the Editor. *Communications of the ACM* **11**(3): 148–149.

Morris, R. (1968). Scatter storage techniques. *Communications of the ACM* **11**(1): 38–44.

Morse, P. L. (1976) January. *User Manual for B1700 SPITBOL; Version 1.0* (Unpublished technical report). Amherst, New York: State University of New York at Buffalo, Department of Computer Science.

Mylopoulos, J., Radler, N., Melli, L., and Roussopoulas, N. (1973). 1.pak: A SNOBOL-based programming language for artificial intelligence applications. In *Proceedings of the Third International Joint Conference on Artificial Intelligence,* pp. 691–696. Stanford, California: Stanford University.

Newsted. P. R. (1975). *SNOBOL; An Introduction to Programming.* Rochelle Park, New Jersey: Hayden Book Co.

Noll, L. W. (1971) April 15. *A Text Formatting Program for Phototypesetting Documents* (Unpublished technical report). Holmdel, New Jersey: Bell Laboratories.

Ophir, D. (1974). *SNOBOL+* (Unpublished technical report). Beer-Sheva, Israel: Atomic Energy Commission, Nuclear Research Centre-Negev.

Osterweil, L. (1970) August 12. *SNOBOL4 Version 3.4 on the UNIVAC 1108 under Exec 8* (Unpublished technical report). Silver Spring, Maryland: Language and Systems Development, Inc.

Pagan, F. G. (1978). Formal semantics of a SNOBOL4 subset. *Computer Languages* **5**(12): 46–49.

Pfeffer, A. S., and Furtado, A. L. (1973). Pattern matching for structured programming. In *Conference Record of the Seventh Asilomar Conference on Circuits, Systems, and Computers, Pacific Grove, California,* pp. 466–469.

Poage, J. F. (1965) September 29. *GE 645 Software Working Group* (Memorandum for File). Holmdel, New Jersey: Bell Laboratories.

Poage, J. F. (1966a) May 13. Letter to R. E. Griswold.

Poage, J. F. (1966b) November 3. Letter to C. Y. Lee.

Poage, J. F. (1977) December 1. Letter to R. E. Griswold.

Puckett, A. L., and Farlow, C. W. (1966) April 28. *Character and Bit String Manipulation Facilities for FORTRAN IV* (Unpublished Technical Memorandum 66-6322-5). Holmdel, New Jersey: Bell Laboratories.

Renne, H. S. (1963) October 1. Letter to F. J. Singer.

Renne, H. S. (1964) August 18. Letter to J. A. Baird.

Ripley, G. D., and Griswold, R. E. (1975). Tools for the measurement of SNOBOL4 Programs. *SIGPLAN Notices* **10**(50): 36–52.

Rodgers, E. A. (1966) August 5. *Symbolic Differentiator and HSUB Compiler Using SNOBOL* (Unpublished Technical Memorandum 66-3241-4). Murray Hill, New Jersey: Bell Laboratories.

Roosen-Runge, P. H. (1967) August. *A Table of Bell Polynomials.* Ann Arbor, Michigan: University of Michigan, Mental Health Research Institute. Communication 212.

Rosin, R. F. (1967). Strings in PL/I. *PL/I Bulletin* No. 4, pp. 6–12. Attachment to *SIGPLAN Notices* **2**(8).

Rosin, R. F. (1969). Contemporary concepts of microprogramming and emulation. *Computing Surveys* **1**(4): 197–212.

Rossman, G. E., and Jones, L. H. (1974). Functional memory-based dynamic microprocessors for higher level languages. *SIGPLAN Notices* **9**(8): 37–65.

Santos, P. J., Jr. (1971) December. *FASBOL, A SNOBOL4 Compiler*. Berkeley, California: University of California, Electronics Research Laboratory. Memorandum No. ERL-M134.

Sears, W. R. (1974) November 25. *The Design of SIXBOL, A Fast Implementation of SNOBOL4 for the CDC 6000 Series Computers*. Tucson, Arizona: University of Arizona, Department of Computer Science. SNOBOL4 Project Document S4D45.

Shapiro, L. G. (1976) March. *Inexact Pattern Matching in ESP*$^3$. Manhattan, Kansas: Kansas State University, Department of Computer Science. Technical Report CS76-10.

Shapiro, L. G., and Baron, R. J. (1977). ESP$^3$: a language for pattern description and a system for pattern recognition. *IEEE Transactions on Software Engineering* SE-3(2): 169–183.

Shapiro, M. D. (1969) March 1. *CDC 6000 SNOBOL4 (Version 2.0) User's Guide*. Lafayette, Indiana: Purdue University, Computer Science Center. Report R0 SNOBOL4-1.

Shapiro, M. D. (1970) December. *An Introduction to Character String Operations Using FORTRAN IV and the Purdue University String Handling Utility Package (PUSHUP)* (Unpublished technical report). Lafayette, Indiana: Purdue University.

Shapiro, M. D. (1972a) June. *A SNOBOL Machine: Functional Architectural Concepts of a String Processor*. Ph.D. Thesis, Purdue University.

Shapiro, M. D. (1972b) September. A SNOBOL machine: a higher-level languge processor in a conventional hardware framework. In *Innovative Architecture, Digest of Papers from COMPCON '72, Sixth Annual IEEE Computer Society International Conference*, pp. 41–44. San Francisco, California.

Silverston, S. M. (1976a) August. *SNOBAT 1.9*. Ames, Iowa: Iowa State University, Computation Center. Technical Report No. 17.

Silverston, S. M. (1976b) December. *Storage Structure and Management in the SNOBAT Implementation of SNOBOL4*. Ames, Iowa: Iowa State University, Department of Computer Science. Technical Report 76-14.

Silverston, S. M. (1977). Extensions to SNOBOL4 in the SNOBAT implementation. *SIGPLAN Notices* **12**(9): 77–84.

Simon, A. H., and Walters, D. A. (1964) December 28. *RCA SNOBOL Programmers' Manual* (Unpublished technical report). Princeton, New Jersey: RCA Laboratories.

Sinowitz, N. R. (1966). Private communication to R. E. Griswold.

Smith, D. C., and Enea, H. J. (1973). *MLISP2*. Stanford, California: Stanford University, Artificial Intelligence Laboratory. Report AIM-195.

Smith, E. (1970) September. *Interactive SNOBOL4*. El Segundo, California: XDS Program Library Catalog No. 890637-11A00.

Sommerville, I. (1977). *S-SNOBOL—Structured SNOBOL4*. (Unpublished technical report). Edinburgh: Heriot-Watt University, Department of Computer Science.

Stewart, G. F. (1975). An algebraic model for string patterns. In *Conference Record of the Second ACM Symposium on Principles of Programming Languages, Palo Alto, California*, pp. 167–184.

Storm, E. F. (1968) CHAMP—character manipulation procedures. *Communications of the ACM* **11**(8): 561–566.

Strauss, H. J. (1968) July 15. *External Functions for SNOBOL4* (Unpublished Technical Memorandum 68-3344-3). Holmdel, New Jersey: Bell Laboratories.

Syrett, T. (1971). *The SNOBOL Machine: The First Virtual Machine Language for the SLAC MLP-900* (Unpublished draft). Stanford, California: Stanford Linear Accelerator Center.

Tamir, M. (1974) August. *Control Mechanisms in SNOBOL* (Unpublished technical report). Jerusalem: Hebrew University of Jerusalem.

Tennent, R. D. (1973). Mathematical semantics of SNOBOL4. In *Conference Record of ACM Symposium on Principles of Programming Languages, Boston, Massachusetts*, pp. 95–107.

Tesler, L. G., Enea, H. J., and Smith, D. C. (1973). The LISP70 pattern matching system. In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pp. 671–676. Stanford, California: Stanford University.

Tharp, A. L. (1977). *Applications of SPITBOL*. Raleigh, North Carolina: North Carolina State College.

Tye, T. T. (1972). *CISBOL; Compiler Implementation of SNOBOL* (Unpublished technical report). Tucson, Arizona: University of Arizona, Department of Computer Science.

Uhr, L. (1968). Private communication to R. E. Griswold.

Uhr, L. (1974) December. *EASEy: An English-Like Programming Language for Artificial Intelligence and Complex Information Processing*. Madison, Wisconsin: University of Wisconsin, Computer Sciences Department. Technical Report 233.

Wade, L. (1970) October 17. *PDP-10 SNOBOL4 User's Guide*. Maynard, Massachusetts: Digital Equipment Corporation. DECUS Program Library No. 10-104.
Waite, W. M. (1967–1973). SNOBOL Bulletins in *SIGPLAN Notices*. (Appearing irregularly.)
Waite, W. M. (1967). A language-independent macro processor. *Communications of the ACM* **10**(7): 433–440.
Waite, W. M. (1969). *The Stage2 Macro Processor*. Boulder, Colorado: University of Colorado, Computer Center. Report No. 69-3.
Wetherell, C. (1978). *Etudes for Programmers*. Englewood Cliffs, New Jersey: Prentice-Hall.
Wilson, D. L. (1966) August. *SNOBOL3*. Milwaukee, Wisconsin: University of Wisconsin, Computer Center. Technical Report.
Wilson, F. C. (1975) April. *A Macro Programming Language*. College Station, Texas: Texas A&M University Graduate Center. NTIS Report AD-A009294.
Wilson, T. C. (1967) July 19. *No. 1 ESS—Special Purpose SNOBOL3 Functions* (Unpublished Engineers Notes). Indian Hill, Illinois: Bell Laboratories.
Yngve, V. H. (1958). A programming language for mechanical translation. *Mechanical Translation* **5**(1): 25–41.
Yngve, V. H. (1964). COMIT (oral presentation). Holmdel, New Jersey: Bell Laboratories.
Yngve, V. H. (1972). *Computer Programming with COMIT II*. Cambridge, Massachusetts: MIT Press.
Zweig, R. (1970) February 17. *FORTRAN Language Extensions for Character String Manipulation* (Unpublished Technical Memorandum 70-9155-1). Holmdel, New Jersey: Bell Laboratories.

## TRANSCRIPT OF PRESENTATION

JAN LEE: Our first speaker is Ralph Griswold. Ralph received his Ph.D. from Stanford in 1962, which, within the period that we are discussing, makes him one of the new kids on the block. At the time of work on SNOBOL languages, begun in 1962, he was a member of the technical staff of Bell Labs in the Programming Research Studies Department at Holmdel, New Jersey. The members of this department were engaged in a variety of research projects related to the use of computers for high-level, nonnumerical computation. The work on SNOBOL was motivated by a need for a better tool to support these research activities. Since that time Ralph has moved to the University of Arizona where he is Professor of Computer Science and the head of the department.

RALPH E. GRISWOLD: The SNOBOL languages constitute a development over quite a period of time, an evolutionary development [Frame 1]. [Frame 2] is a time line. The solid lines represent periods of active language development; the circles represent specific re-

1. The Setting
2. Original Goals
3. SNOBOL Characteristics
4. Goals for SNOBOL2 and SNOBOL3
5. SNOBOL3 Characteristics
6. Goals for SNOBOL4
7. SNOBOL4 Characteristics
8. Contributions of the SNOBOL Languages
9. The Future of SNOBOL

**Frame 1.** The SNOBOL languages.