AB26.2. CORRESPONDENCE

AB26.2.1. Retained data-objects

18 Ebury Street,
London S W 1.

May 12th 1967

To: Members of WG2.1 and Editor, AB.

An essential feature of ALGOL60 was the transience of all data to the scope - extent in time - of the block in which it was declared.

Thus the notion of data whose scope is more extensive than the program that created or referred to it is outside the universe in which an ALGOL program lives and dies.

The purpose of the letter is to bring to the attention of WG2.1 the need to be able to manage such data in universes in which we can hope to be able to code in ALGOL67. I think specifically of parallel processes cooperating and competing through a core of retained data-objects.

The following are the facilities I see required.

1) The ability to create retained objects.
2) The ability to control the later access of foreign processes to the retained objects.
   a) To read
   b) To alter
   c) To execute
   d) To destroy
3) The ability of foreign processes to acquire access rights to retained objects.
4) The ability of processes to assume temporary constancy or exclusive access to retained objects and the sequencing of processes making conflicting demands.

I do not believe that the semantics of what is required is yet sufficiently well-understood for WG2.1 to lay down the syntax in which these abilities should be expressed; however I believe it to be timely to admit to ALGOL67 retained objects, leaving open for future work specification of the commands that enable them to be managed.

J.G. Laski

supported by M. Woodger

AB26.2.2. Features essential for a workable ALGOL X

[The following letter from D.T. Ross to A. van Wijngaarden and B.J. Mailloux is reproduced here at the request of its author, May 19th 1967. -Ed.]

Electronic Systems Laboratory,
Massachusetts Institute of Technology,
Cambridge, Mass. 02139.

October 20, 1966.

My primary activity since my return home last Friday, (except for sufficient time to plow through the correspondence which had accumulated during my absence), has been to try to collect and make explicit my suggested modifications to Warsaw-2 [a working document of WG2.1 - Ed.] for possible incorporation into the next draft of ALGOL X. As usual, it has been very difficult to keep features which will be appropriate for ALGOL Y (or perhaps ALGOL Z) from intruding, but I think I can summarize here for you those features which I feel are not only appropriate but necessary for a successful ALGOL X if it is to be more than a prettied-up version of some of the features which now exist in our AED-0 System with a few additions. I hope ALGOL X can be much more.

As was apparent from the Warsaw meeting, the crucial question concerns the explicit use of references or pointers in the language, and the implications of this on the dynamic handling of free storage. In our present AED-0 system, these features are quite explicitly present in raw form, and the net effect is that the sophisticated user can do almost anything he desires, but there are numerous pitfalls for the average user. Although AED-0 is supposedly a high-level compiler language, being an extension of ALGOL60, nonetheless, it is very much at the level of detailed, free-wheeling machine-code programming with respect to the subject matter. I have been making the analogy that FAP is to FORTRAN as AED-0 is to a true high-level language for plex programming. I would like to see if we could make ALGOL X be much more of a high-level, reliable, pitfall-free language, at least for a proper subset capable of doing the basic record handling. I think it is unreasonable to expect the mass of potential users to become sufficiently sophisticated to work properly with anything less.

Although I have not yet plowed back through the material in Warsaw-2 for a re-check, I believe your description of the <u>level</u> process is exactly the appropriate mechanism for the internal operation of any implementation of this sort of system. Therefore, in the suggestions that follow, there is no attack on the reference concept itself, but only an attempt to hide these delicate matters in the <u>sub rosa</u> workings of the system. The intent is to save the user from a great deal of intricate, complex, repetitive thinking, and to protect him from subtle and very-easy-to-make errors.

Principle features

In order to completely eliminate the concept of reference from the language itself, it is necessary to have the entire system based upon the concept of manipulation of "objects", where an object may be an integer, a real-number, a "person", or a variable of the program, or a variable whose value is a variable of the program, etc.. The only way in which an object can be accessed or manipulated is by assigning it to be the value of some appropriate variable or field. There can be no unassigned objects in limbo. The system then is concerned with objects, variables, and procedures which act on objects which are values of variables.

In addition to the ability to define new types of objects as cartesian projects of already-defined objects, it is necessary to group various types of objects into classes, (i.e., group sub-classes into classes to use Tony Hoare's terminology) so that a powerful, but fully-controlled and constrained descriptive mechanism results. With this mechanism it is possible to assign objects as values to variables or to fields of records. With ALGOL Y or Z it will not be necessary to be explicitly concerned with how this assignment takes place, but at this point in time it appears to be necessary to have a new word "copy" in the language in order to indicate when a new and distinct version of an object is to be created and assigned. Furthermore, although for ALGOL Y this would not be necessary, for ALGOL X if the object is an entire plex structure, i.e., a whole network of inter-connected records, copying of such structures must be performed by an explicitly-written procedure.

Finally, although its necessity is possibly debatable, an extremely natural and useful adjunct to the entire system is a preset or initialization facility, so that the initial values of all variables can be explicitly controlled. This also allows elaborate initialized plex structures to be incorporated directly into a program by the compiler rather than having program statements which laboriously generate the same initial structure each time the program is begun.

These then are the features which I feel are necessary for ALGOL X if it is to fulfill its intended role. There are many other topics which are natural extensions of the above basic ideas, but these can appropriately be held for the successors to ALGOL X. I now will elaborate on each of the above features.

Class and Type Definitions

Following the examples of Hoare's summer session notes (pages 9 and 16) but changing vocabulary words and sequence so that the notation for calling a record is more parallel to the definition, we have the following notation. To define a new type of record:

> type person (integer date of birth; boolean male; (person) father,
> elder sibling, youngest offspring);

Then T may be declared to be a person variable by writing

> (person) T;

after which a particular person may be created as the value of T by

> T := person (1908, true,,,);

(The empty argument positions are equivalent to Tony's null.)

In order to define a class of types we write:

```
class expression is constant (real value) ora variable (string printname)
      ora class pair is (sum ora difference ora product
          ora quotient)((expression) left operand, right operand,
              derivative);
```

after which we may declare new variables that are of type expression, constant, pair, and product

```
(expression) X; (constant) Y; (pair) Z; (product) Q;
```

and create values to be assigned to those variables.

```
X := Y := constant (10.5);
Z := Q := product(variable('ALPHA'), constant(2.), constant(2.));
```

In general if A is an identifier of a class name, B and C are types, D through M are field declarations such as "real value", etc., and D′ through M′ are actual expressions of the corresponding types, in general: we have

```
            may be none       types       may be none
class A is (D,E,F)(B(G,H) ora C(I,J))(K,L,M);
            X := B(D′,E′,F′,G′,H′,K′,L′,M′)
                                                 calls
            Y := C(D′,E′,F′,I′,J′,K′,L′,M′)
```

Another feature which is quite essential if reasonable implementations are expected, is some means for specifying the size of various fields so that multiple fields may be packed into a single word on implementation. This could be done either in terms of ranges,

```
type person (integer date of birth from 1850 to 1966, ...
```

or number of bits

```
type person (integer 13 bits date of birth, ...
```

or best of all, by giving a transformation function to permit efficient encoding:

```
type person (integer 7 bits (date of birth - 1850), ...
```

(It probably would be best to require that both the transformation and its inverse be provided explicitly for storing and retrieving values of the field.) This transformation feature may perhaps be an ALGOL Y feature, as is the concept of "coupled" or "constrained" fields in which formulas are given as part of the record definition, which automatically set certain fields to be functions of values in other fields. An example would be a geometric vector record in which in addition to fields specifying the end points, another field contains the length of the vector. You will recall our brief discussions in Warsaw about the difficulty of thinking of algebraic operations on records to produce other records. I believe the concept of coupled fields is a necessary component of any such scheme (probably ALGOL 7).

Parameter Mechanism

The key concept for a parameter machanism which is consistent with the whole set of remarks in this letter is that when a procedure is called, the actual parameters of the call provide objects which are used as values wherever the formal parameters appear in the procedure definition. In other words, the problem of reference versus non-reference which consumed so much of our time in the Warsaw discussion is primarily a matter of implementation, but as far as the user of the language is concerned, all actual parameters are objects. In order to obtain the reference kind of actual parameter, then it is necessary merely to enable the language to refer to the variables of a program as objects. The introduction of a new vocabulary word, var , allows the introduction of a meta language level in which program variables are treated as objects in the same way as person objects, or real or integer objects. Thus for example the procedure "inner product" of Section 12.2.2 of Warsaw-2 A might begin by:

let real proc innerproduct (int var i, int n, real proc xi,yi) be ...

Note the parallelism between "int var" and "real proc". This is really the point of it all. As far as the user of the language is concerned, he has objects, variables, and procedures with which he may work. Example 12.2.3 would read

let real proc inprod (real array [1:n] a, b) be ...

because an array is itself a suitable object. (I.e. the use of ref here in Warsaw-2 is redundant when arrays are objects.) Note that since arrays always must be arrays of some type of thing, I recommend moving the word real to come first. Also the fact that n is an integer need not be explicitly stated.

Note that full type control as described in the previous section applies to all formal parameters of procedure definitions, so that when a procedure is called, it is necessary merely to write the expression or identifier which specifies the object being transmitted.

In the proper subset of ALGOL X, the word var would be permitted only as illustrated above in the specification of formal parameters, whereas in the full ALGOL X, with its "user beware" potential pitfalls, it would be possible to declare variable-valued variables such as

int var X; int var var Y;

I.e., an integer variable may be assigned as value of X, and X, being an integer-variable variable, is suitable as a value for Y.

Zones, Declarations, and Copies

As I tried to explain several times in Warsaw, there is a fundamental distinction between zone structure which concerns the physical space of existence of objects, and block structure which concerns the scope of identifiers. Zone structure and block structure are complementary, and may in fact be confused and identified if objects are restricted to single word lengths as in ALGOL60, but they must be recognized as distinct as soon as "large" objects, such as records, arrays, and multiple long values, and plex structures are considered.

Otherwise efficient implementation becomes impossible, and the richness of the language suffers. Although I suspect that in ALGOL Z (or perhaps it should be ALGOL omega) zone structure could itself be treated entirely sub rosa, and only as part of the implementation, for ALGOL X it seems to be necessary to be aware of it when the features of the language are described. I do not propose that explicit zone terminology be included in ALGOL X, although it should be included in ALGOL Y because it yields much more elegant and efficient problem representation in many cases. For ALGOL X, however, it seems sufficient only to be concerned with zone structure as it relates to the block structure.

Another simplification for ALGOL X is to continue to treat definition and declaration as one and the same thing, although as I tried to describe in Warsaw, I believe the distinction is very important for ALGOL Y. It is in preparation for such distiction that the above examples of procedure definition begin with "let real proc" rather than merely "real proc". When the distinction is made, we can declare and define a procedure separately or together (a feature which is needed for separate compilations and useful in other ways as well).

real proc inprod(...	declaration
let inprod be	definition
let real proc inprod(...) be ...	definition and declaration

For ALGOL X only the latter case would be allowed for procedure, type, and class definitions. (In AED-0 we now say define... to be instead of the shorter let .... be .)

A variable or field of a given type can only be declared within the scope (in the block structure sense) of that type's definition, for ALGOL X. The simplest implementation would also create all records for objects of that type in the zone associated with the definition block. Since the previous suggestions provide complete type and class constraint, however, it appears to be a fairly straightforward task for an implementation to associate with each record generator call (which I prefer to think of as a call on the record type itself, without the concept of a generator being involved) the outermost zone to which actual assignment of that object need be made. Even though a type may be defined in the outermost block, an individual record of that type may only be used as value (however indirectly) in an inner block, in which case the zone of the inner block should be the place of residence of that particular object.. The compiler can quite easily trace through all possible assignments in which that object appears on (or participates in) the right-hand side to determine the required zone.

Although as I mentioned before, ALGOL Y or Z may not require it, ALGOL X appears to require the vocabulary word copy in order to specify when a fresh copy (using the record generator) should be made for assignment as value. The idea is that each unique object exists in the zone structure exactly once, so that an explicit copy command is required if duplicate copies are desired. If X is a person variable, then

Y := copy X;

would merely assign to Y a duplicate copy of the person record (possibly in a different zone, if Y is subsequently assigned in a more restricted scope than that of X). In a similar way, Z := <u>copy</u> X (1906, F, , ,);

would make Z be a female person born in 1906 with the same father, elder sibling and youngest offspring as X. (In copying, a null space specifies no change in value.) Note that copy only will copy single records. Copying of entire record structures would have to be done by an explicit program written by the user.

For ALGOL Y I will propose that the concept of type be augmented to include entire plex structures rather than merely single records. Thus for example, a plex of type family could be defined in terms of the father, sibling, and offspring fields of person-type records. Then entire families can also be manipulated as single objects. A family might be defined by some such notation as

<u>plex</u> family <u>is</u> person (, , not null, not null, not null);

in which the notation "not null" indicates that in the process of recursively copying or otherwise scanning to determine the boundaries of a plex of type family, the values of those components are to be elaborated as long as they are not null, i.e., a null value in one of those fields would terminate the recursive elaboration. A richer terminology than this is required for the full job, however.

As to the destruction of records, the zone structure coupled with block structure provides the basic mechanism, but considerably more efficient schemes may be implemented by putting a reference count in the physical record itself and destroying that record whenever the count becomes zero. Or one can do a standard LISP type garbage collection (at much greater expense than in LISP, however, due to the more complex data structure). It may also be possible to have the compiler automatically insert individual destruction commands at appropriate places on the basis of the scan of usage mantioned above.

<u>Preset Facility</u>

A final feature for ALGOL X concerns augmenting the constant declaration feature of Warsaw-2 by the ability to <u>preset</u> or initialize the value of any type of variable. For example:

<u>real</u> pi := 3.14159;

permits one to write the short identifier "pi" instead of the longer string of digits, but the "pi" always has the same constant value.

<u>preset</u> <u>real</u> X := 3.14159;

on the other hand would make X be a real variable whose initial setting is the indicated value. Other values may, however, be assigned to the variable X in the operation of the program.

This feature becomes particularly important in the case of record variables since anexpression such as

<u>preset</u> T := person(1908, <u>true</u>,,, person(1937, <u>false</u>, T, J:person
(1935, <u>true</u>, T, person(1934, <u>true</u>, T,,),
person(1964, <u>true</u>, J,,),);

would create at one whack [ouch!] the entire plex structure on page 11 of Hoare's summer notes. Note that J is a label attached to a particular call for a person record, and used only within the preset statement range for establishing the necessary connection. The person variable T which is actually a variable of the program can serve the same role within the present statement.

In general, preset blocks may contain arbitrary declarations and variables, so that the right-hand sides of preset statements may be arbitrary expressions in the language. Preset blocks obey the same block structuring as the same program itself by implication. For example

```
begin
            preset begin
              ---
            end preset
            ---
      begin                              nested
            preset begin
            ---
            end preset
      end
end
```

so that a full but independent block-structuring is available for the right-hand sides of <u>preset</u> statements. Identifiers appearing within a preset block but not declared within the preset block structure are variables from the program block structure.

Conclusion

These then are the features which I feel at present are essential for a workable ALGOL X. I am indeed sorry that I must present these ideas to you in this informal fashion, but my profound unfamiliarity with the fine rigor of Warsaw-2, coupled with the fact that I already have expended much more time than I should in thinking through these various ideas, makes it impossible for me to attempt a rigorous formulation. Even if I had the time to spend, I suspect it would take me so long to achieve facility with your techniques that the transmission of the ideas themselves would be delayed far too long to be of any use. My impression in Warsaw was that the actual incorporation of most of these ideas into the modified Warsaw-2 would not be terribly difficult for one who knew how to go about it, so that I hope you will be able to follow through on the incorporation of these ideas into ALGOL X.

I am sure that the presentation here contains some obscure points, and I hope that you will feel free to bombard me with questions, complaints, and suggestions to further clarify the concepts.

Douglas T. Ross,
Head, Computer Applications Group.

AB26.2.3. Comment on AB25.3.2.

3375 Alma Street, Apt. 354,
Palo Alto, California 94306.

June 5th 1967

In AB25.3.2 I.D. Hill discusses the for statement proposal of Wirth and Hoare which treats the controlled variable as a dummy implicitly declared local to the for statement. Hill's claim that this notion destroys the copy rule concept for procedures with name parameters is in error. The controlled variable can simply never be affected by application of the copy rule.

It is true that Hill's Program 1 does not in the Wirth and Hoare language perform the sum that it does as an ALGOL60 program. But neither does Hill's Program 2 result from his Program 1 by applying the copy rule, if the controlled variable is a dummy. Application of the copy rule to Program 1 results in a program with the same performance, and so no conflict with the copy rule as a semantic device arises.

Larry L. Bumgarner

AB26.2.4 Collection of tested algorithms

Chelsea College of Science
and Technology,
Computer Centre,
Manresa Road, London SW 3

August 1967

I am enclosing a list of our current holding of tested algorithms. It would be most helpful if you could draw AB readers' attention to our collection, and our desire to enter into exchange arrangements.

As a general rule we test algorithms for theoretical correctness, limitations and performance. A master file is then written which indicates the scope and use, with examples. I would welcome hearing from people interested in a collaborative effort in this field, and we would put them on our mailing list for new material and changes. Of course we would expect a reciprocal arrangement !

Our current efforts are confined to ALGOL versions of algorithms, but we would also be delighted to receive material in other languages with a view to its hand translation into ALGOL and subsequent evaluation.

Richard F. Shepherd,
Director.

[The current list contains 349 algorithms from sources including CACM, Regnecentralen Copenhagen, Stanford University, Mathematisch Centrum Amsterdam, NPL, TH Munich, ... - Ed.]

AB26.2.5 Integer Representation for ALGOL Basic Symbols

General Electric Company
Information Systems Equipment
13430 N. Black Canyon Highway
Phoenix, Arizona 85029.

9 August 1967

AB25.3.4 noted with much interest. This problem has been considered for some time, and in a sphere somewhat larger than that evidenced by the contents of the Bulletin. For example, the minutes of the 1959 May 1 meeting of the SHARE IAL Committee state:

> "Green (Julien): Remarked that the group (IBM) had decided on an internal representation using 8 bits for IAL symbols (256 character set)."

This stemmed from an earlier paper (enclosed) from me to that body on 1959 February 18, which states "let us ignore the exterior symbols until SHARE or the Committee can compromise suitably.....The Reference Language.... contains at least 106 symbols. Let each of these be represented internally by bit groups of a length to be determined by the fabricators..."

I also enclose my document ISO/TC97/WG E(USA-11)56, Stockholm meeting 1962 May 9, Programming Language Implications on Coded Character Sets, which reads:

> "For example, single characters for ALGOL word symbols.....It is proposed that a tentative group of characters (to follow the escape characters) be reserved to indicate special alphabets for programming languages. Further, work should begin at once on a specific alphabet for interchange of ALGOL and COBOL programs...It is advisable that a generally agreed upon nucleus of graphic assignments (e.g., common to the ECMA, BSI and ASA draft codes) be retained in the programming set. Also those ALGOL and COBOL graphics defined in any of these sets should be preserved."

I trust this gives me sufficient antiquity to say that your correspondents are all way off base in their proposals, as far as specific assignments are concerned. How anyone could propose seriously an adaption of ATLAS or KDF9 codes when the above-mentioned interchange code is in its final stage as an ISO Draft Standard, corresponds to CCITT Working Alphabet No. 5 and the USA Standard (and perhaps many other national standards) is quite beyond me. The character set people worldwide have also, for some time, decided against any checking of the Hamming type in the coded representations.

My purpose is not to complain. I want to ensure that:

1) The proposals in the AB and any momentum that might ensue are in correspondence with international standard codes for information interchange, since what is proposed falls in this category.

2) The character set people in ISO, ECMA, and national bodies learn that the matter is being discussed seriously.

3) The programming language people in ISO, etc., are aware that they had better get around to following my 1962 proposal, and

4) The ISO code must be an invariant subset of the ALGOL interchange code.

R.W. Bemer

Copies to:

D. Hekimi, ECMA
P.B. Goodstat, BEMA
L.L. Griffin, USASI X3.2
A.C. Grove, USASI, TC97 Secretariat
T.B. Steel, Jr., USASI X3.4

[First enclosure]

February 18, 1959

Mr Frank Engel
Chairman, IAL Committee
SHARE

Dear Frank:

Since your committee has had difficulty in agreeing upon hardware representation to implement IAL for the 704 and 709, I make the following proposal:

1. The main difficulties of hardware representation are those created by external communicative equipment. No difficulties exist internally for the 704 or 709 (binary) machines.

2. Let us ignore the exterior symbols until SHARE or the Committee can compromise suitably. This may be deferred for a reasonable period of time without damaging or setting back any effort to create processors.

3. The Reference Language should be construed as the ideal language for internal manipulation. It contains at least 106 symbols. Let each of these be represented internally by bit groups (or bites) of a length to be determined by the fabricators of the processor, with the restriction that they should be the lowest binary representations in these groups. It should be borne in mind that, for internal purposes, variables and identifiers may all be represented by simple single symbols, as Professor McCarthy has pointed out. The only restriction here shall be that the basic symbols of the IAL shall form a contiguous set.

   To allow for possible new operator symbols, at least the lower 256 or 512 representations should be reserved. An obvious corollary of this is that those IAL symbols which exist in the Navy collating sequence (as given for the 705 computer) should be preserved in that internal sequence. A certain flexibility of assignment should be reserved to match future computers with eight-bit representations.

4. The processors should have many other design criteria but for purposes of standard dissemination the format of the exchange medium should be a binary tape (or the equivalent thereof) containing programs in the internal representation. This will allow one or many hardware representations as SHARE pleases or as individual installations may prefer.

5. Obviously all processing from original card input must be done in a single module which has as its only function the creation of the binary tape mentioned in Item 4, in an unambiguous mapping from the external symbols on a one-for-one basis.

R.W. Bemer

[Second enclosure]

International Organization for Standardization — ISO/TC 97/WG E (USA-11) 56

9 May 1962

TECHNICAL COMMITTEE 97 — COMPUTERS AND INFORMATION PROCESSING

Working Group E — Programming Languages

Reference: ISO/TC 97/WG E (USA-3) 22.

Subject: Programming language implications on coded character sets

Gentlemen:

A. The following points of consideration are proposed as a basis for study of character sets, as proposed by WG B and others, with regard to their suitability from a programming language viewpoint:

1. Are the sets capable of graduated size by means of regular and clearly defined rules of expansion ?

2. What sizes are of particularly concern to programming language needs ?

4 - BIT	16
5 - BIT	32
6 - BIT	64
7 - BIT	128
8 - BIT	256
others ?	

3. What is the preferential ordering of graphics for inclusion in sets of varying size ?

4. Are there natural orderings of Sub Groups of graphics which should be reflected in the coded representations ? Are there artificial orderings which have caused unnecessary constraints ?

5. Are control characters the concern of programming languages, as well as information characters ?

6. Are there defined correspondences between multiple character symbols used with smaller sets and the single characters available in larger sets ? Is there a controlled pattern of replacement with increasing set size ? For example, single characters for ALGOL word symbols, or multiple symbols such as <- for := , ^ for / \.

7. Are criteria specified for controlled future expansion and addition ?

8. Is an "escape character" provided to allow for usage of alternate, specialised character sets ? What graphic subsets should remain in common positions in alternate sets so that they may be uniquely recognisable ?

9. What values following the escape character (see section B of this document) should be reserved for specialized character sets for programming languages ?

10. What control considerations are necessary to utilize alternate or graduated equipments ?

11. Are sufficient data delimiters included in sets (of sufficient size) to adequately indicate set membership in data groups ?

12. Are sufficient data delimiters included in sets (of sufficient size) to adequately indicate syntactic grouping ?

13. What characters are required to map two-dimensional flowcharts into one-dimensional string languages ? For example, a character signifying "is inside of" and characters for the shaped elements and flow lines.

14. Are the character sets equally applicable to conventional von Neumann machines and streaming (or other types of non-von Neumann) machines ?

B. From the [proposed] IFIP definition of "escape character" it may be seen that it is possible to have many compatible alphabets within the framework of an international charecter set and code. The imminence of such a standard interchange code and the existing problem of interchange of algorithms suggest that the time is proper for some advanced planning in the area of characters used in programming languages.

It is proposed that a tentative group of characters (to follow the escape characters) be reserved to indicate special alphabets for programming languages. Further, work should begin at once on a specific alphabet for interchange of ALGOL and COBOL programs.

It is advisable that a generally agreed upon nucleus of graphic assignments (e.g., common to the ECMA, BSI, and ASA draft codes) be retained in the programming set. Also those ALGOL and COBOL graphics defined in any of these sets should be preserved. Consideration should then be given to adding the rest of the ALGOL characters, in particular reserving positions for the "complete word" characters of ALGOL. Among the further graphics which may be considered are those likely to have special usage stemming from mathematics and logics, delimiters which imply set membership hierarchy, delimiters for syntactic entities such as phrases, flow chart symbols, etc..

R.W. Bemer

Howard Bromberg
U.S. Representative
ISO/TC 97/WG E