SETL Newsletter # 171B

J. Schwartz
A. Grand
December 14, 1976

Remark on the Implementation of The Basing Scheme.

This newsletter will outline what seems to be an efficient scheme for implementation of the fundamental rules for basing semantics described in Newsletter 171A.

Every object will contain two fields which collectively determine its basing. These are

i. a FORM field, which contains an index to a table called the *forms table*.

ii. a BASEARRAY field, which points to the start of an array of pointers to the various bases of the object.

The forms table entry indexed by an object's FORM field will give the length of the BASEARRAY for the object.

The BASEARRAY pointer for a subobject X of an object O can point into the corresponding array for O. (For example, this is likely to happen when O is a set and X one of its elements, or when O is a multi-valued map and X is one of its domain sets.) This 'overlapping' style of pointer use resembles a technique which can be used to handle long character strings, and is equally unproblematical.

Basearrays can be held in vectors of an appropriate special form, which the garbage collector can handle using a technique like that presently used for strings. New basearrays need only be built at those (relatively rare) points at which a base assignment takes place. Let $b_1,\ldots,b_n$ be the bases involved in a base assignment, and let $m_1,\ldots,m_k$ be all the declared (formal) repr's in which these bases are involved. Then at the point of the base assignment the compiler can interpolate an operation which generates a new 'comprehensive base array' A, which is long enough for the base arrays required for every one of the modes $m_1,\ldots,m_k$ to appear as a contiguous subarray of A. If we assume also that the current values of all declared repr's are held in an auxiliary 'declared repr list', so that $m_1,\ldots,m_k$ are represented by entries in this list with indices $r_1,\ldots,r_k$, then we must also update each of these entries by changing the BASEARRAY pointer which it contains. If this approach is used, then (declared) repr's can passed to run-time routines which need them simply by passing an appropriate repr index r.

Let us used the term *full form* for a repr list entry, i.e., a pair (F,P) consisting of a form index and a BASEARRAY pointer. Given the full form for a composite object (set, map, or tuple) the full form for any particular subobject (member, domain or range element, component) will be calculatable simply as $(\phi(F), P + \psi(F))$, where $\phi(F)$ and the offset $\psi(F)$ are values held in appropriate fields in the forms array entry referenced by F.

To test the validity of an undeclared-to-declared assignment D = G, one checks the form of G for equality with the known form of D, and the base-list pointer held in G for equality with the current base list pointer for the declared repr list known to the compiler. This check can be made rapidly by a 'convert nubbin' which calls the general convert routine when the check fails. The first action of the convert routine can be to check for equality of form, and if this holds to compare the base array of the object to be converted with the base array of the specified target repr. If these two arrays have identical components, then no conversion is necessary.

In an operation like D = B with C, C will be converted by the with routine to have the repr appropriate for elements of D. Again, conversion can be avoided if C already has this form.

If D has general repr, then no conversion is necessary; but it is well for the with routine to check the full form of C for agreement with the full form of elements of B, so that form information is not unnecessarily lost when B is built A similar remark applies to other incorporation operations such as D(B) = C.