SETL Newsletter 183

E. Schonberg
R. Dewar
A. Grand
J. Schwartz
December 16, 1976

Some Revisions of Basing Semantics and Implementation.

1. It is often a useful and inexpensive to maintain two or more representations of a single object. Accordingly, we allow multiple repr's to be stated for a single object. The suggested syntax is illustrated by

$$s:\underline{set}(<\in b_1,<\in b_2,\in b_3>>),\ \underline{map}\{\in b_1\}\underline{map}\{\in b_2\}\ \underline{set}(\in b_3);$$

The implementation of this is unproblematical; the compiler simply generates additional variable names, and assigns a single repr to each of these names. In our example s would be fragmented into two names s_1, s_2; source operations changing s would be compiled into corresponding changes to both s_1 and s_2. In expanding an operation that used but did not modify s, the compiler could choose to use either s_1 or s_2 as input to the expanded operation; the object form leading to the most efficient code would be used. Similarly, operations incorporating s into a larger object will choose the most effective of s_1 and s_2 for incorporation. Assignment of s to a variable g of type general will be implemented as an assignment of one of s_1 and s_2 (perhaps always the first) to g.

2. The implementation of the present b_2:base($\in b_1$) construct will be modified so that, whereas a field for a pointer to an element block of b_1 will always be reserved in each element block of b_2, this field will not be filled in until some reference to an element block eb_2 of b_2 attempts to access this field. When such an access is attempted, the required element block eb_1 of b_1 will be located by hashing (and inserted into b_1 if necessary), and the field in eb_2 which points to eb_1 will be filled in.

When this is done, the value pointer in eb_2 may also be modified to match that in eb_1. In the special case in which an object known to have $\in b_1$ format is inserted into b_2, its eb_2 field may be filled in at once.

An advantage of this scheme is that it lowers the cost of initial insertion of eb_2 into b_2. This allows us to base b_2 on more than one other base, much as if we wrote b_2:base($\in b_1$, $\in b_3$,...). However, since we may also wish to declare a repr constra

$$b_2:(b_1, b_3,\ldots)\ \underline{\text{base}}(mode)$$

is to be preferered.

Note that this scheme allows 'circular' constructions such as

(*) $\quad b_1:(b_2)\underline{\text{base}}\ (\underline{\text{int}}),\ b_2:(b_1)\underline{\text{base}}\ (\underline{\text{int}})$

which might for example create a base and a subbase which point to each other. In this way, 'plexes' efficient for certain purposes can be created. Note that if a construction like (*) is used, we can fill in pointers from b_1 to b_2 whenever pointers from b_2 to b_1 are filled in, and vice-versa.

3. The former construction s:set($\in$b) is now perceieved as redundant, since much the same effect can be achieved by writing b_2:(b)base, s:subset(b_2). This change also has the beneficial effect of speeding up iteration over s. Thus we will drop the set-of-elements construct. This makes the syntax set($\in$b) that we formerly used for set-of-elements available for what was formerly written as subset(b). Note that each element block in a base will have a few bits available for the storage of local subset indicators.

If a base b supports only a small number $s_1,\ldots,s_2$ of local subsets (but no maps and no elements with $\in b$ basing other than iterators over local subsets based on b) then there will exist no pointers to completely null element blocks of b. In this case, the NELT field of the header of b can be used to keep count of the number of totally null blocks which the base contains; this count must be updated whenever a destructive deletion operation is applied to some s_j. At the start of each iteration this count can be compared to the hashtable size of β, and if the number of null element blocks is excessive the base can be rehashed. By proceeding in this way, the density of null element blocks can be held down to something in the neighborhood of 50%.