XEROX
PALO ALTO RESEARCH CENTER
Computer Sciences Laboratory
INFORMATION PRODUCTS GROUP
System Development Division
July 7, 1977

DRAFT - DRAFT - DRAFT

To: A Sampling of Mesa Users

From: Ed Satterthwaite, John Wick

Subject: Variant Record Changes

Keywords: Mesa, Variant Records

Filed On: <WICK>VRCHANGES.BRAVO

In a recent meeting of the Mesa Language Working Group, we considered two changes in the implementation of variant records that might impact existing code in adverse ways. We would appreciate your comments on these proposals soon, so they can be included in the upcoming release definition (due July 10). Please pass this memo on to anyone in your group who might like to review it.

Variant Record Packing

The current algorithm for packing fields into multi-word records right justifies (and widens, if necessary) the rightmost field in a word. For example, if the last field in the record requires five bits, but eight bits remain in the word, the value of that field will be stored right justified in the eight bits. This allows more efficient code in some cases. More importantly, records do not have any empty holes (filler fields) in this scheme. Thus records can be compared without worrying about garbage values in fill fields. (Record comparison is a useful operation, and field by field comparison is an unacceptably ugly alternative for either the user or the compiler.)

The tag fields of variant records currently do not follow this rule; the tag is left justified in the remaining space in the word. This allows variants that begin with small fields to "eat into" the remaining space. It has the disadvantage that most variant records cannot be compared, because of the possibility of a garbage fill field to the right of a tag.

The proposal is to convert to the right-justifying scheme for variant record tags. The advantage is that comparison could be allowed. The disadvantage is that some variants that formerly occupied n words would now take $n+1$ words, since there would be no fill field to "eat into". Here's a worst case example (the notation to the right indicates field positions as offset:first-bit..last-bit inclusive):

```
Foo: TYPE = RECORD [                    Old         New
  f1: [0..37777B),                      0:0..13     0:0..13
  f2: SELECT * FROM                     0:14..14    0:14..15
    red => [b: BOOLEAN],                0:15..15    1:0..15
    blue => [c: CARDINAL],              1:0..15     1:0..15
    ENDCASE]
```

Note that, under the old scheme, bit fifteen of word zero of a blue Foo is unused (garbage), hence blue Foos cannot be compared. On the other hand, in the new scheme, both variants would occupy two words.

Note also that, under the new scheme, the tag field would not be widened if there were some field in *each* variant that could be used to fill the remaining space. Thus if the blue variant of Foo included some 1-bit field, a red Foo would occupy one word again (at least if the record were not mutable; see below). If, as in this case, the variants were of differing lengths, the compiler would still require discrimination before comparison.

Mutable Variant Records

There is a reasonably well known bug in the language definition that allows the type of a variant record to be changed "on the fly", often with disasterous consequences. Consider the following code (assume the definition of Foo above):

```
foo: POINTER TO POINTER TO Foo;
baz: POINTER TO Foo;

WITH record: foo↑↑ SELECT FROM
  red =>
    BEGIN
    foo↑↑ ← Foo[3, blue[5]];
    IF record.b THEN . . . ;
    END;
  blue =>
    BEGIN
    foo↑ ← baz;
    . . . record.c + 1 . . . ;
    END;
  ENDCASE
```

There are two problems here. In the red arm, the storage containing the discriminated record is overwritten with a blue Foo. Subsequent references to fields of a red Foo then make no sense. Furthermore, the storage for the original red Foo might have been obtained by requesting SIZE[red Foo] words from an allocator. If so, the store clobbers the (logically unrelated) word that follows the allocated storage. In the blue arm, the access path foo↑↑ is changed so that subsequent mentions of record reference a different Foo, possibly with the wrong variant.

The *Early Experience with Mesa* paper contains a discussion of these problems. We don't know of a solution to the second problem (short of copying on discrimination) that is compatible with the variations on compacting allocators that are so popular, and we intend to ignore it. We believe that the first problem can be alleviated, however, and it seems desirable to protect users (and ourselves) against the sort of memory clobber that it allows. We would like to implement the first (and potentially

incompatible) steps of that solution as soon as possible.

The solution is to attach to the declaration of a variant record an attribute indicating whether the record is *mutable* or not. The idea is that, unless the record is mutable, the tag field is set upon initialization and never changed. Allocators can provide exactly the amount of storage required for the particular variant of an immutable record, but we adopt the convention that storage allocated for a mutable record must be large enough to hold *any* variant.

To guarantee that an immutable tag is not changed in a sneaky way, the rules for pointer assignment, etc., must also change: assignments of the form

```
x: POINTER TO red Foo;
y: POINTER TO Foo;
y ← x
```

become illegal unless Foo is immutable (otherwise, the new pointer could be used to change the tag so that x would no longer point to a red Foo). With this change, discriminating on an immutable record is safe with respect to tag changes; overwriting a discriminated (or undiscriminated) mutable record is still possible, and there will always be enough space to avoid clobbering unrelated storage.

Other advantages are possible in future versions of Mesa. For example, a proposed addition is an ALLOCATE expression intended to subsume a common but well understood loophole and to eliminate most uses of SIZE. With the conventions outlined above, allocation of variant records could be handled in a consistent way and, e.g, a NEXT operator upon pointers into sequences of variant records could be defined.

Unfortunately, the current version of Mesa lacks special syntax to indicate the initialization of dynamically allocated storage, and the initial solution would be incomplete. The short term (next release) consequences of this change thus would be the following:

Adding a specification of the mutability to the declaration of variant records.

Enforcing the pointer assignment rule given above.

Changing the definition of SIZE (to take mutability into account).

Requiring the declaration of an immutable variant record (as a local variable or record field) to specify the particular variant, either explicity or implicitly (by initialization).

The last of these is negotiable; because of the problem of distinguishing initialization, actual protection of an immutable tag from overwriting by assignment will not be possible until a later release.

The immediate question is whether mutable or immutable should be the default case. The keyword VARYING has been suggested to indicate mutable records, or we might choose CONSTANT to indicate the immutable property (both would follow the SELECT keyword in the record declaration). We are currently leaning toward the VARYING attribute for compatability reasons, since the current implementation is closer to immutable records, except for the assignment check. Which do you prefer?

Distribution

Prototype Software: Malloy, Simonyi
Juniper Project: Morris, Sturgis
Tools Environment: Ayers, Smokey
Communications: Crowther, Murray
APL Project: Wyatt

Geschke
Johnsson
Koalkin
Lampson
Mitchell
Sandman
Sweet
Weaver