SETL Newsletter # 130B

J.T.Schwartz
May 30, 1975

The Use of Equalities in the Deduction of Inclusion/Membership Relations.

1. Confirmation of relationships without use of chains of equalities.

In this newsletter we shall discuss the inclusion/membership analysis algorithms described in NL 130, justifying their correctness, and emending a number of inaccuracies concerning the use of equalities in inclusion/membership analysis.

Note first of all that in writing a relationship oRox, we assert that immediately after o has been evaluated it has a value *valo* which stands in the relationship R to the value *valox* calculated at the last evaluation of ox prior to the evaluation of o. (Assuming that o and ox are not the same ovariable, this is the value which ox retains when valo is calculated.) Similarly, in writing iRox, we are asserting that at the moment of its use i has a value equal to that last calculated for ox. The assertion essential to justification of the 'elimination of relationships' method of inclusion/membership deduction sketched in NL 130 is that any set of relationships confirmed by this method of deduction (for brevity, we shall call these 'confirmed relationships') is true.

For this assertion to be justified, we must define our deduction method clearly, and restrict it carefully in one particular regard. The situations which make necessary the restriction to which we allude are typified by the following example:

```
s = ...
s' = nl;                 /* line 2 */
(while ...)
     s = s less y;       /* line 4 */
     x = ∋s;             /* line 5 */
     s' = s' with x;     /* line 6 */
end while;
```

In this code, the ivariable occurence of s in line 5 is linked only to the ovariable occurence os of s in line 4. Thus we can be sure that ox $\in$ os (where ox is the ovariable occurence of x in line 5.) The ivariable occurence of s' (line 6) is linked only to the ovariable occurences of s' in line 2 and 6, which we shall call os2' and os6' respectively. Since os2' $\ni\in$ os (because the value of os2' is <u>nℓ</u>) it might appear that there was no reason ever to eliminate the plausible relationship os6' $\ni\in$ os, yet as a matter of fact this may well be false since s is diminishing, perhaps to <u>nℓ</u>, while s' is increasing. The trouble comes from the fact that line 4, which modifies s, can be executed between the time that os6' is calculated and the time that its value is used.

This makes it plain that our deduction algorithm should not confirm a relationship iRox if there exists an ovariable o $\in$ud(i) and a path from o to ox to i free of occurences of other ovariables o'$\in$ ud(i). Setting aside all special issues involving the use of equality (these issues will be discussed later in the present newsletter) we can state the rules to be applied in this case, together with a number of other significant supporting definitions and rules, as follows:

A. Relationships iRox and oRox can be confirmed either on *value grounds* or on *standard grounds*.

B. A relationship iRox (resp. oRox) is confirmed on value grounds (which we will abbreviate as *vconfd*) if either:

a. constant values are known for i and ox (resp. o and ox) and the relationship R is seen to hold for these constant values; or

b. a constant value is known for i (resp. o) and R is seen to hold in view of this known value and the known type of the value of ox (here an example would be i = <u>nℓ</u> and ox a set, in which case i $\ni\in$ ox can be vconfd); or

c. a constant value is known for ox, and R is seen to hold in view of this known value and the known type of the value of i (or o). (An example here would be ox = <u>nℓ</u>, in which case we can be sure that i $\ddot{\in}$ ox is true.)

C. A relationship oRox will be confirmed on standard grounds (which we will abbreviate as *sconfd*) under the conditions explained in NL 130, i.e., if appropriate relationships i_jRox involving the argument ivariables i_j of o are confirmed. A relationship iRox will be sconfd if the relationship oRox is confirmed for each o ∈ ud(i), and if, whenever oRox is sconfd rather than vconfd, there can exist no path from o to ox to i which does not pass through some other variable in ud(i).

D. Cases in which o and ox are the same ovariable require special treatment, and are probably best handled by not admitting any relationship of the form oRo as plausible unless it is time *a priori*.

Given these rules, it is not hard to see that every confirmed relation is true.

To prove this, we envisage some run of the program P which we are analysing, consider the full sequence of ovariable evaluations which takes place during this run, and let the n-th evaluation in this sequence evaluate o.

We argue by induction on n. If n = 1, then o must be set either by a read statement, in which case the set of confirmed relationships oRox will be null, or o must be set from a constant, i.e., from an ivariable whose value is known, and then clearly each confirmed oRox must be vconfd. But it is plain that all vconfd relationships are true.

Now suppose that n > 1, and first consider the case of a confirmed relationship iRox involving one of the argument ivariables of o. If vconfd, this relationship is true. Otherwise it is sconfd. The value of i used in evaluating o will be that stored at the last preceeding time that an ovariable o' ∈ ud(i) was encountered. Since iRox is confirmed, o'Rox must also be confirmed, and hence either vconfd or sconfd. If o'Rox is vconfd, then o'Rox must remain true when i comes to be used, even if the value of ox has changed since o' was evaluated, since if ox can change, o'Rox must hold (as a relationship between ovariable values) by virtue of the known value of o' and the type of ox. If o'Rox is sconfd, then by rule (c) above, the path from o' to i cannot have passed through ox. By inductive hypothesis, o'Rox was true (as a relationship between values) at the moment that o' was evaluated; since the value of ox cannot have changed, iRox must remain true (as a relationship between values) when i comes to be used. And now, since oRox is by assumption sconfd, it is, when regarded as a relationship between ovariable values, a logical consequence of relationships involving argument ivariables, which relationships are known to be true. Hence oRox is true for $n > 1$ completing our induction and proof.

The following is a practical technique for imposing the restriction that a relationship iRox should not be sconfd unless there exists no $o \in ud(i)$ and path o to ox to i along which no other $o' \in ud(i)$ is encountered.

i. Ignoring this restriction, generate a preliminary estimate of the set of all confirmed relationships.

ii. Form the set of provisionally confirmed relationships iRox for which there exists an $o \in ud(i)$ such that ox can be reached from o along a path clear of occurences of the variable v common to o and i.

iii. For each such relationship, modify the text of the source program being processed by inserting an assignment v = v into it and re-analyse data flow. If after this the ovariable of this assignment appears in ud(i), then iRox must be dropped.

iv. After applying rule (*iii*) to drop some collection of relationships iRox, proceed, much as in step i, to eliminate additional relationships until a mutually confirming collection is obtained. By the preceeding proof, all the relationships which remain must necessarily be true.

2. The use of chains of equalities.

Next let us consider relationships of the special form o eq ox, and the way in which the preceeding argument is changed if we allow reasoning by chains of equalities.

Note first of all that, in the present context, the relationship oy eq ox is not symmetric. In writing oy eq ox, we assert that immediately after the evaluation of oy, oy has the same value as was last calculated for ox; in writing ox eq oy, we assert that immediately after the evaluation of ox, ox has the same value as was last calculated for oy. Suppose now that ox eq oy has been proved, and that we also know that oy cannot appear on a path from ox to o that does not go through ox twice. Let valox (resp. valoy) be the value obtained when ox (resp. oy) was last calculated prior to some particular calculation of o. Let valoy' be the value obtained when oy was last calculated prior to the calculation of valox. Then since by assumption the value of oy is not recalculated between the caluclation of valox and the calculation of o, valoy and valoy' must be the same. Thus the relationships oRoy and oRox are equivalent. To fix our attention on this useful fact, we state it formally as a lemma.

Lemma 1. Let ox eq oy be true, and suppose that oy cannot appear on a path from ox to o that does not pass through ox twice. Then if oRox is true, so is oRoy, and vice-versa.

Next suppose that o' eq o, and that ox cannot appear on a path from o to o' which does not go through o twice. Let valo be the last value calculated for o before some particular evaluation of o', and let valox be the last value calculated for ox before valo is calculated. Then at the moment of calculation of o', valox is still the last value calculated for ox. Hence if oRox is true, then o'Rox is true. Suppose next that o' eq o,and that ox cannot appear on a path from o to o' which does not go through o' before reaching o again or reaching a program exit node. Then the value valo calculated for o at some given moment is equal to the value valo' calculated for o' when o' is next encountered; and between these two calculations neither valo nor the last previously calculated ox value valox will not change.

Hence if o'Rox is true, then oRox is also true. The following lemma summarises these observations.

Lemma 2. Let o' eq o be true, and suppose that ox cannot appear on any path from o to o' that does not go through o twice. Then

i. If oRox is true, then so is o'Rox.

ii. If o'Rox is true, and if in addition every path starting at o must pass through o' before it reaches o again or reaches an exit node, then oRox is also true.

It is easy to give examples which show that the hypotheses appearing in Lemma 1 and 2 are essential. First consider the code

```
s = ...                         /* line 1 */
s' = nl;                        /* line 2 */
(while ...)
     s = ...                    /* line 4 */
     if ... then quit;;
     s' = s;                    /* line 6 */
end while;
t = s' less ...;                /* line 8 */
```

Denote the ovariable occurences of t, the two ovariable occurences of s' (in lines 2 and 6), and the two ovariable occurences of s (in lines 1 and 4) by ot, os2', os6', os1, and os4 respectively, and the ivariable occurences of s and s' by *is* and is'. Then *is* is linked only to os4, so os6' eq os4. Moreover is' is linked only to os2' and os6', and since os2' $\ni\in$ os6', we have ot $\ni\in$ os6'. But ot$\ni\in$os4 need not be true, since os4 can be re-evaluated between the execution of line 6 and the next following execution of line 8.

As a second example related to Lemma 1, consider the code

```
(while ...)
     s = ...
     if ... then quit;;
     s' = s;                          /* line 4 */
end while;
t = s;                                /* line 6 */
```

Let the ovariable occurences of s, s', and t be called os, os', and ot respectively, and let the two ivariable occurences of s (in lines 4 and 6) be called is4 and is6 respectively. Then since is4 is linked only to os, we have os' eq os. Similarly, ot eq os. But ot eq os' can clearly be false.

Next we give an example showing that if its hypotheses are substantially relaxed Lemma 2(i) may cease to be true. Consider the code

```
sx = ...                         /* line 1 */
sy = nl;                         /* line 2 */
(while ...)
     s = sy less ...;            /* line 4 */
     sx = sx less ...;           /* line 5 */
     sy = sx;                    /* line 6 */
     s' = s;                     /* line 7 */
end while;
```

in which o- and ivariables osx1, osx5,osy2, osy6, os,os', isy, isx5, isx6, and is occur (the reader will easily identify these occurences.) Since isx6 is linked only to osx5, osy6∋∈osx5. Since osy2∋∈osx5 also (by vconfirmation), we have os∋∈osx5. Clearly os' eq os; yet os'∋∈osx5 may be false since sx can change (by the execution of line 5) after s is calculated (in line 4).

Finally, we give a simple example which shows that the second part of the hypotheses of Lemma 2(ii) cannot be substantially relaxed. Consider the code

```
x = ...
y = ...
if y ∈ x then
     y' = y
else ...
```

which may also be written

```
x = ...
y = ...
if y ∈ x then
          y = y oralternatively ∋x;
          y' = y;
else ...
```

Then it is clear that oy' eq oy and that oy'∈ox is true; however there is no reason why oy ∈ox should be true.

If we substitute an ivariable i for the ovariable o' in Lemma 2, we obtain a statement which is also true. To see this, let the variable of the ivariable i be v, introduce an assignment vv = v immediately before the occurence of i, and let the resulting ovariable occurence of vv be called o'. Then plainly iRox is equivalent to o'Rox for all ox, while paths to i and paths to o' are essentially the same.

Equality relationships should be used in the following way to deduce additional relationships of membership and equality for a program P. We begin by calculating the class $CREL_1$ of all confirmed (i.e.,vconfd and sconfd) relationships for P without making any special use of equality relationships. By the argument presented in section 1, all these relationships are true.

Some of the relationships in $CREL_1$ may be relationships of equality. By applying the principles embodied in Lemma 1 and 2, these relationships can be used to confirm a still larger set $CREL_1'$ of relationships. Specifically, given a relationship oRox in $CREL_1$ or $CREL_1'$, we

i. Add oRoy to $CREL_1'$ if oy eq oy and there is no path from ox to oy to ox which does not go through ox twice;

ii. Add oRoy to $CREL_1'$ if oy eq ox and there is no path from oy to ox to o which does not go through oy twice;

iii. Add o'Rox to $CREL_1'$ if o' eq o and there does not exist a path from o to ox to o' which does not go through o twice;

iv. Add o'Rox to $CREL_1'$ if o eq o' and if in addition every path starting at o' must pass thru o before it reaches o' again or reaches an exit node.

It is clear from Lemmas 1 and 2 that all the relationships in $CREL_1'$ are true. Next, using these relationships, and proceeding as in section 1, we can generate a still larger family of relationships $CREL_2$. This is done as follows: we extend the definition of the term 'sconfd' by including any relationship oRox in $CREL_1'$ in the set of confirmed relationships; then $CREL_2$ is the set of all relationships which are vconfd or sconfd in this extended sense. The family of relationships $CREL_2$ can be extended to a larger family $CREL_2'$ in much the same way as $CREL_1$ was extended to $CREL_1'$ and then a set $CREL_3$ can be derived from $CREL_2'$ etc.

A few relationships which would remain out of reach if no special use was made of relationships of equality can be derived in the manner just explained. As an example, consider the code sequence

```
s = ...;
s' = {x ∈ s|...};
y = <y,s'>;
s" = {x ∈ s'|...};
u = y(2);
x = ∋s";
```

Here we have oy 2 eq os', so that ou eq os'; and os"∋ ∈ os', from which it follows that os" ∋ ∈ ou belongs to $CREL_1'$ (but not to $CREL_1$), and that ox ∈ ou belongs to $CREL_2$. On the other hand, consider the sequence

```
s = ...;                           /* line 1 */
ss = nl;                           /* line 2 */
(while ...)
     s = s less ...;               /* line 4 */
     x = ∋s;                       /* line 5 */
     y = <y,x>;                    /* line 6 */
     u = y(2);                     /* line 7 */
     ss = ss with u;               /* line 8 */
end while;
```

Here ovariables os1, os4, oss2, oss8, ox, oy, and ou, and ivariables is4, is5, iy6, iy7, ix, iu, and iss occur (the reader will readily identify these occurences.) It is readily seen that ox ∈ os4, so that oy2 ∈ os4, and thus ou ∈ os4 and oss ∋∈ os4 all can be confirmed without any special use of equality relationships becoming necessary.

An inclusion/membership analysis algorithm may or may not decide to make special use of equality relationships; it is not at all clear from the preceeding examples that it is worth while doing so. If these relationships are exploited, it will be necessary to find all cases in which o' eq o, and in which relationship oRox, o'Rox, oxRo or oxRo' holds, and where there also exists a path from o to ox to o' not going through o twice. This can be done with reasonable efficiency as follows: for each pair of ovariables such that o' eq o is confirmed, find the set S_{from} (o) of all blocks which lie along a path from o, and the set S_{to} (o') of all those blocks which are the origin of a path to o' not going through o.

Then the ox which belong to $S_{from}(o) * S_{to}(o')$ and which are related to o or o' are the ones we want.

In connection with Lemma 2(ii) we will want to find pairs o', o such that o' eq o and such that there exist paths from o which encounter either an exit node or o again before they pass thru o'. Paths of this kind can be found by an analog of the live/dead analysis algorithm.

3. More complex equality relationships.

Beside simple equality relationships o eq ox, we can consider more complex relationships oη eq ox, or even oη eq η' ox. The operators which can reasonably appear in η are the component operators n and perhaps also ∞ and $\bar{n}$; only component operators n can reasonably appear in η'. Analogs of Lemmas 2 and 3 can be stated for these more general cases:

Lemma 3: (Analog of Lemma 2). Let o' η eq o be true, and suppose that ox cannot appear on any path from o to o' that does not go through o twice. Then

i. If oRox is true, then so is o' η Rox;

ii. If o'η Rox is true, and if in addition every path starting at o must always pass through o' before it reaches o again or reaches an exit node, then oRox is also true.

To prove Lemma 3(i) first suppose that η is a sequence of component operators: $\eta = n_1 \ldots n_k$. Let valo be the last value calculated for o before some particular evaluation of o' which yields the value valo', and let valox be the last value calculated for ox before valo is calculated. Then at the moment of evaluation of o', valox is still the last value calculated for ox. Hence if oRox is true, we have valoRvalox, and thus valo' $(n_1, \ldots, n_k)$ R valox, i.e., o'η Rox.

Since o ∞ Rox is equivalent to o $\bar{1}$ Rox, and since o $\bar{n}$ Rox simply means that o mRox for all $m \geq n$, Lemma 3(i) is proved not only for sequences of component operators but also for sequences of operators of the form n, $\bar{n}$, and ∞.

Next consider Lemma 3(ii), first supposing that η is of the form $n_1, \ldots, n_k$. Suppose that the hypotheses of Lemma 3(ii) are satisfied. Then the value valo calculated for o at some given moment is equal to valo' $(n_1, \ldots, n_k)$, where valo' is the value calculated for o' when o' is next encountered; and between these two calculations neither valo nor the last previously calculated ox value valox will change. Hence o'ηRox implies oRox; and Lemma 3(ii) follows immediately.

Lemma 4 (Analog of Lemma 1). Let ox η eq oy be true, and suppose that oy cannot appear on a path from ox to o that does not pass thru ox twice. Let $\eta = n_1, \ldots, n_k$ be a sequence of component operators, and let $\tilde{\eta} = n_k, \ldots, n_1$. Then if oR $\tilde{\eta}$ ox is true, so is oRoy, and vice-versa.

To prove this, let valox (resp. valoy) be the value obtained when ox (resp. oy) was last calculated prior to some particular calculation of o. Then by hypotheses valox $(n_1, \ldots, n_k)$ is the same as valoy, and thus oR $\tilde{\eta}$ ox and oRoy are equivalent.