SET Newsletter #35 May 13, 1971

New form for IFF-statement Dave Shields

The iff-statement consists of a header and a trailer. The header consists of the keyword iff, followed by a series of iff-elements, each of which (in the simpliest form of iff-statement) is either a test-node, consisting of a name followed by a '?' sign, or an action-node, consisting of a name followed by a ',' sign. To the lower-left of any test-node follows its positive-case descendant, and to the lower right follows its negative-case descendant. The deepest-rightmost descendant in the header must be an action-node consisting of a name followed by a ';', the ';' indicating the end of the statement header.

The trailer section of the statement defines the structure of the nodes contained in the header. A definition for the node nam begins with a statement having nam as its label, and consists of either

a) all following statements in the trailer up to but not including the next labeled statement, or
b) all statements up to but not including the statement with label lab, in which case the definition must begin with

name: til lab; ... .

The iff statement is terminated by either a repeated semicolon, or "end iff", or "end iff token;".

Each node-definition defines the "value" of the node; the last statement in the sequence of statements defining the node statement may have either of the forms

= expn; or to name;

where expn is an arbitrary SETL expression and name is the name of a node in the tree described in the header.

The final statement of a test-node must have the form "=expn". If the value of expn is true, control will pass to the left descendant of the test node; otherwise, control will pass

to the right-descendant. The final statement of an action-node may be of the form "to name", where name is a node in the tree; this node is the successor-node to which control will pass after executing the node-definition. If a statement of this form is omitted control will pass to the next statement following the iff-statement.

If an action node in the tree is not defined in the trailer, then the action implied is a branch to whatever statement, in the subroutine containing the tree, which has as its label the node name. For example, in

```
s1: ...;
s2: ...;
tree: iff test?
          actl,s1;
test:= i eq j;
actl: ...;  end iff; ...
```

control will pass to statement s1 if the value of test is not true. The occurrence of a node name in the header which is neither defined in the trailer nor corresponds to a statement in the program is considered a syntactic error.

Subnodes. The trailer may contain definitions for nodes not occurring in the header. Such definitions define sub-nodes, which have a value of the form '=expn'. Any reference to a subnode nam in the definition of another node of the tree is treated as though nam were a variable whose value is the value of the node nam. This allows the programmer to defer the definition of parts of a node.

For example, consider the print routine for SETL, in which we print an object by indenting a few spaces (except in the case that this is first object output during this call), inserting the number of the object, a period, and then the printed form of the object. This subpart of the program may be expressed as

```
printnextojb:  iff firstobj?
                    printit, indentprint;
firstobj := ncalls eq 0;
indentprint :  indent(somespaces); to printit;
printit :  addprintrep;
somespaces := levelofobject * spacesperobject;
levelofobject := level(object);
spacesperobject := 3; end iff;
```

Note that somespaces is a sub-node referred to by indentprint; and that somespaces refers to the subnodes levelofobject and spacesperobject; thus the argument to indent, somespaces, has as value "level(object)*3".

The names which label the nodes in the iff-statement are assumed to be known only within the statement; that is, one cannot enter the tree in a nonstandard way by executing goto node; where node is a label in the trailer of an iff-statement.

We now describe options allowing more flexible constructions:

a) the nodes in an iff-statement header may be replaced by the code defining their values, if the code is enclosed in parentheses. For example, we may write

```
tree:  iff (x gt 0) ?
           (y = x+1; to on;),(y gt 0)?
                              on, (z=x+y;);
on:    subr(x,y);  end iff;
```

b) any action node may be preceded by an iteration-header. By this we mean that the code in the node definition is to be executed over the "iteration set" defined by the iteration header. When the iteration set is exhausted, the successor node is determined in the usual manner. For example, we may write

```
iff (set ne nl)?
    (∀ s ∈ set(cls))doelem,printullcase;
doelem: ...;
printnullcase: nerrs= nerrs+1; print(errmessage);
  if nerrs gt maxerrs then exit;;
errmessage := 'error-since null set'; end iff;
```

c) We define a composite node to consist of a name followed by the sign "+", and to have a single descendant node (which may itself be composite) written immediately below it. The definition of a composite node nam may not contain any value-statement; i.e. no statement of either of the forms "=expn;" or "to expn" is legal in its definition. For example consider

```
iff t1?
    act1+, t2?
    act3, act1, act2;
act1: ...;  t1:= ...; t2:=...; act2:...; act3:...
end iff;
```

If t1 has value true, we do action act1, and then action act2; if t1 is not true we test t2 and then do either act1 or act2.

d) Test nodes may have more than two descendants. Such nodes, called multi-test nodes, have the form

```
multi? k
d1,d2,...,dk ;
```

Here k is an integer constant ($k \geq 3$) and the k nodes d1,...,dk are the descendants of the node multi. Each descendant may in turn be a test or action node, but no two descendants may have the same name. The node multi must be defined in the trailer by an iff-statement which has among its action nodes the set of descendants di. We speak of this latter iff-statement as an imbedded iff-statement; the trailer-elements for the imbedded iff-statement are mingled in any order with the trailer-elements for the iff-statement in which it is imbedded.
For example, consider

```
start: iff multi? 3
          casel, case2, case34?
                        case3, case4;
          case34  := not count3 lt count4;
          multi : iff model?
                      casel, mode2?
                             case34, case2;
          casel:
          case2:
             model := ...;
             mode2 := ...;
          case3:
          case4: ...; end iff multi;
```

When control reaches start, we first evaluate the multi-test. If, for example, model is not true and mode2 is true; then we exit from the iff-statement defining the multi-test with case34 as the exit-node.
Returning the enclosing iff-statement, we see that case34 is in fact a test which must be evaluated to find which of actions case3 or case4 is to be done, etc.

e) If the name nam occurs only once in the header or in any imbedded iff-statement as a test- or multitest-node, i.e. only one instance of nam is followed by the sign '?', then nam may also be used as an action node. The implied intention is to transfer the test named nam and then select the descendant in the usual way. This gives a 'looping' effect. For example, the SETL while statement

(while c doing bb)b; end while; next:

may be expressed as the iff-statement

```
iff nc?
    nb+ quit
    nbb+
    nc;
nc := c;
nb := tℓ nbb; b;
nbb : b; end iff;
```

Note here that the special node name quit may be used to refer to the first statement after the iff-statement.

f) For clarity, an exit-node may be preceded by the word "to". For example consider

```
s1:
   ...
s2: iff t1?
        actl, to s1;
    actl: sub(x,y); end iff;
s3: ...                          .
```

In evaluating the iff-statement labeled s2, if t1 has value true, then we call sub(x,y) and leave the tree, continuing at s3; if t1 does not have value true, we exit to statement s1 in the program.

ifx-expression

The ifx-expression is defined to provide the same two-dimensional style for the conditional expression that the iff-statement provides for the if-statement. The ifx-expression has the same syntax as the iff-statement, except that an action node may contain a value statement. Such a node may not contain a successor statement, i.e. a statement of the form "to node;". The value of the ifx-expression is the value obtained by evaluating expn in the first action-node processed whose definition ends with "=expn;". For example, consider

```
x =  ifx (a gt amax)?
           printerr + (=a)
         (=amax);
printerr:  print 'error a too large'; end ifx;;
```

Here we assign a to x unless a exceeds some value amax, in which event we print an error message and assign amax to x.

Note that the ifx-expression, like the standard conditional expression, may occur on the left-hand side of an assignment statement. Also, any subnode of an iff-statement may have its value defined by an ifx-expression.