An Algorithm for Use-Definition Chaining:

Several forms of optimization depend on knowing which definitions in a program can affect the environment at a given point in the control flow graph of a program. By a definition of a variable x, we mean an assignment of a value to x. This newsletter presents an algorithm which computes the set, \text{reaches}(b), of all definitions of variables for which there is a definition-clear path to the entry to block b. This algorithm uses the interval technique and might be considered the "dual" of the dead-variable analysis algorithm (SETL Newsletter Number 28).

For the purposes of the analysis, we need the following sets and functions.

1. \text{defs} - the set of all definitions in the program.
2. \text{var}(\text{defn}) - the function which maps a definition onto the variable it defines.
3. \text{thru}(b, sb) - the set of variables for which there is a definition-clear path through b to sb.
4. \text{def}(b, sb) - the set of definitions in b from which there is a definition clear path for the variable defined to an exit from b to sb.
5. \text{initial} - the set of initializing definitions made before entry to the program.

The following functions are assumed to have been provided by the interval analysis.

1. \text{s}(b) - the set of immediate successors of b.
2. \text{p}(b) - the set of immediate predecessors of b.
These functions are also defined on all intervals of the derived graphs.

The first step in our analysis is to calculate \textit{thru} and \textit{def} for an interval, given these sets for the nodes of the interval. To do this, we will first compute three intermediate sets.

1. \textit{path}(b) - the set of variables for which there is a definition-clear path from interval entry to \textit{b}.
2. \textit{defint}(b) - the set of definitions in the interval which can reach \textit{b} by any path not including a latch (a branch back to the head).
3. \textit{defhead} - the set of definitions in the interval which can reach the head by a latch.

We now state some equations involving these sets. Suppose \textit{sb} is the head of some successor interval \textit{sint} of the interval \textit{intv} that we are processing. A definition clear path through \textit{intv} to \textit{sint} must pass through some predecessor of \textit{sb} in \textit{intv} and through that predecessor to \textit{sb}; the SETL code fragment is

\begin{equation}
\text{thru(intv, sint)} = \mu: \text{bep(sint(1))int tl [intv]} \\ \text{(path(b int thru(b, sint(1))}}
\end{equation}

where \textit{intv} and \textit{sint} are SETL sequences (in interval order) of nodes.

Computing the set \textit{def(intv, sint)} for the interval \textit{int} is more complicated. A definition within \textit{intv} can reach \textit{sint} if it is in one of two sets.

1. \[\mu: \text{bep(s intv(1))} \ (\text{def(b, sint(1))} \ \mu \ \{	ext{d\in defint(b) | var(d)\in thru(b, sint(1))}\});\]
   - the set of definitions in \textit{b} with a def-clear path to
sint(1) and definitions in intv which reach the entrance to b and pass through b to sint(1).

2. \( \{ \text{defhead} \mid \text{var}(d) \in \text{thru}(\text{intv}, \text{sint}) \} \)

- the set of definitions which reach the head of intv via a latch and whose variables have def-clear paths through intv to sint.

Therefore,

\[
\text{(2)} \quad \text{def}(\text{int}, \text{sint}) = \{ \text{defhead} \mid \text{var}(d) \in \text{thru}(\text{int}, \text{sint}) \}
\]

\[
\cap [u : b \in p(\text{sint}(1))] (\text{def}(b, \text{sint}(1))) : \{ d \in \text{defint}(b) \mid \text{var}(d) \in \text{thru}(b, \text{sint}(1)) \};
\]

The equation for \( \text{path}(b) \) is the same as the one in dead variable analysis.

\[
\text{(3)} \quad \text{path}(b) = [u : pb \in p(b)] (\text{path}(pb) \cap \text{thru}(pb, b));
\]

where

\[
\text{(4)} \quad \text{path}((\text{int})(1)) = \{ \text{all variables} \} = \text{var}[\text{defs}];
\]

The set \( \text{defint}(b) \) is computed by examining each predecessor of b in the interval. A definition will be in \( \text{defint}(b) \) if it is in \( \text{def}(pb, b) \) for some predecessor pb of b, or if it is in \( \text{defint}(pb) \) and its variable is in \( \text{thru}(pb, b) \).

\[
\text{(5)} \quad \text{defint}(b) = [u : pb \in p(b)] (\text{def}(pb, b)) \cup \{ d \in \text{defint}(pb) \mid \text{var}(d) \in \text{thru}(pb, b) \};
\]

where

\[
\text{(6)} \quad \text{defint}(\text{int}(1)) = \text{nl};
\]

The form of these SETL code fragments suggests that we process the nodes of the interval in interval order. Since we need
information concerning the predecessors of each node processed.

The routine \texttt{inout(intervals)}, has, as its only argument, a

sequence of intervals, starting with the intervals of the control

flow graph followed by intervals of the first derived graph and

so on. Each interval in this sequence is a sequence of its nodes

in interval order. For each interval, \texttt{inout} computes the sets,

\texttt{thru} and \texttt{def}, assuming that they are available for the nodes

of the interval. Their availability is assured by the order of the

intervals in the sequence \texttt{intervals}.

\begin{verbatim}
define inout (intervals); optimizer external s,p,thru,def,var,defs:
    /* process each interval */
    (1$\leq$j$\leq$#intervals) intv=intervals(j);
        defint(intv(l))=nl; path(intv(l))=var[defs];
    /* pass through intv to get path and defint */
    (2$\leq$i$\leq$#intv) b=intv(i);
        path(b) = [u: pb $\in$ p(b)](path(pb) $\cap$ thru(pb,b))
        defint(b) = [u: pb $\in$ p(b)](def(pb,b) $\cup$
                        \{d $\in$ defint(pb) $|$ var(d) $\in$ thru(pb,b)\});
    /* compute defhead */
        defhead = [u: pb $\in$ (p(intv(l)) $\cap$ thru(intv,l))](def(pb,intv(l))
                        $\cup$\{d $\in$ defint(pb) $|$ var(d) $\in$ thru(pb,intv(l))\});
    /* compute thru and def for interval */
    (\forall sint $\in$ s(intv)) sb = sint(l);
        thru(intv,sint) = [u: pb $\in$ (p(sb) $\cap$ thru(sb))]
                        (path(pb) $\cap$ thru(pb,sb))
        def(intv,sint) = \{d $\in$ defhead $|$ var(d) $\in$ thru(intv,sint)\}
                        $\cup$ [u: pb $\in$ p(sb)](def(pb,sb) $\cup$
                        \{d $\in$ defint(pb) $|$ var(d) $\in$ thru(b,sb)\})
    end $\forall$sint; end $\forall$i; return; end inout;
\end{verbatim}
This routine is all we need for the first pass. It will calculate \texttt{thru} and \texttt{def} for intervals of the control flow graph and all derived graphs.

The second pass must calculate \texttt{reaches(b)} for every block in the program. A definition reaches \texttt{b} if, for some predecessor \texttt{pb}, it is in \texttt{def(pb,b)} or if it reaches \texttt{pb} and the variable it defines is in \texttt{thru(pb,b)}.

(7) \[ \texttt{reaches(b)} = \left\{ u: pb \in \text{p(b)} \right\} \left( \text{def(pb,b) \cup \text{d} \in \text{reaches(pb) \mid \text{var(d)} \in \text{thru(pb,b)}} \right) \]

For a program entry \texttt{e},

(8) \[ \texttt{reaches(e)} = \text{initial}; \]

and for interval heads,

(9) \[ \texttt{reaches(intv(1))} = \texttt{reaches(intv)}; \]

At the end of the first pass the \texttt{reaches} set for the single node representing the entire program is set to \texttt{initial}. Then the routine \texttt{outin} is called.

\begin{verbatim}
define usedef(intervals); optimiser external p,r,thru,def,defs
initial, reaches, var; inout(intervals);
reaches(interval(#intervals) = initial;
outin(intervals); return; end usedef;
\end{verbatim}

The subroutine \texttt{outin} calculates the \texttt{reaches} set for the nodes of each interval, given this set for the interval itself. It processes the outermost interval first, then the next outermost intervals and so on, passing through intervals in reverse order. Within an interval, nodes are processed in interval order using equation (7).
define outin(intervals); optimizer external s,p,thru,def,var,
    reaches;
    */ process intervals in reverse order */
    (intervals>=1) intv=intervals(i);
    reaches(intv(1))=reaches(intv);
    */ process nodes in interval order */
    (intv<=b<intv) b=intv(i);
    reaches(b)=[u: pbεp(b)](def(pb,b) u
        \{d ε reaches(pb) | var(d) ε thru(pb,b)\});
    end \forall i; end \forall i; return; end outin;

On exit from this routine, \texttt{reaches(b)} will have been computed
for every block \(b\) in the program, which was the desired result.
Note that the form of equation (\ref{equation}) forces us to process in interval
order if we wish to always have the required \texttt{reaches} sets.