SETL NEWSLETTER # 201

Micha Sharir
February 16, 1978

On Compaction on RC-Paths

If space and time required for manipulation of RC-paths is expected to create significant problems, we can use an approach which has not been sketched previously, which will enable us to drastically reduce the size of RC-paths, and to make their manipulation quite rapid. The present note will sketch this approach. Specifically:

After call-graph analysis, we can introduce a new data-flow problem, which might be termed RC-path analysis. Its purpose is to compute all possible RC-paths in the program, compact them in an appropriate way, and pre-compute certain operations of combination between them, as needed in the later phases of the optimizer.

This analysis requires the following inputs:

(1) The call graph (CALLPATHS), as computed by the call graph analysis.

(2) The set of internal paths (INPATHS), presently used in data-flow analysis. This is defined as a set of pairs [I1,I2], where I1 is an entry or a call, I2 is a call or an exit, and there exists an intra-procedural execution path from I1 to I2.

Our algorithm will compute a map REACH, defined on all entries and call instructions. For each such instruction I, REACH(I) is the set of all RC-paths which describe some execution path terminating at I. A recursive equation for REACH is:

(1) $$\begin{aligned} REACH(I) = \{RCP : & [I1,I,F] \in CALL\ PATHS, [I2,I1] \in INPATHS, \\ & RCP_0 \in REACH(I2) \text{ and } RCP := RCP_0 || F \neq \underline{errorpath}\} \\ + \{F \quad : & [I1,I,F] \in CALLPATHS\} \ \underline{with}\ \underline{nullpath} \end{aligned}$$

(This equation is justified in much the same way as the equation used to compute BFROM.) If N denotes the maximum number of cyclic repetitions that we shall later need to trace within any execution path (in the current optimizer, N is the nesting level limit of types), then the RC-path concatenation yields errorpath if a component repeats itself more than N times in the concatenated path.

As already noted in other similar contexts, equation (1) can be solved in a straight forward way by an iterative propagation algorithm.

Next, let f be a compacting map on the set of all RC-paths, mapping each such path to some integer in W := $[0,1\ldots 2^k-1]$, for some small number of bits k. It will be quite advantageous to have $f^{-1}\{0\} = \{$nullpath$\}$ and let us assume also that f(errorpath) is undefined.

For each operation OP between RC-paths we will compute a (possibly multi-valued) map OPCOMP :WxW → W, and for each relation R between RC-paths we compute a relation RCOMP ⊂ WxW, as follows:

```
OPCOMP := RCOMP := nℓ; RCPATHS := range REACH;
(forall RCP1 in RCPATHS, RCP2 in RCPATHS)
        w1 := f(RCP1); w2 := f(RCP2);
        RCP := OP(RCP1, RCP2); w := f(RCP);
        OPCOMP {[w1, w2]} with w;
        if R(RCP1, RCP2) then RCOMP with [w1, w2]; end:
end forall;
```

An appropriate repr for these new objects is as follows:

B_1 : base (int $(0\ldots 2^k-1)$);
B_2 : base ([εB_1, εB_1]);
RCOMP: local set (εB_2);
OPCOMP: local map (εB_2) remote set (εB_1);

If we use these reprs, the space required to store RCOMP and OPCOMP should be very modest, since in effect we are using bit-matrix representations.

After this preparatory phase, we proceed with optimization, using W as the set of all (compacted) RC-paths, and the RCOMP's and OPCOMP's as (pre-calculated) RC-path operations. The only effect of this on optimization algorithms suggested earlier, is that some previously single-valued operations (such as maximum and concatenation) can now be multi-valued, so that slight modifications of the subsequent algorithms will be required.

Remarks:

(1) It may be advantageous to either combine RC-path analysis with the computation of BFROM, or perform RC-path analysis after BFROM has been calculated, for two reasons:

(a) Since BFROM is used heavily in all subsequent optimization algorithms, we wish to compute it very precisely, using non-compacted RC-paths.

(b) Parts of the mechanism needed to perform RC-path analysis are used in the computation of BFROM.

Note, however, that in this approach, after BFROM has been calculated, we ought to replace the RC-paths in it by their compactions.

(2) Some algorithms, such as copy optimization and name splitting, do not require all the RC-path information that might be stored in BFROM, but only need to know the last call or entry instruction through which a BFROM link has materialized. However, there may not be a well defined way to retrieve this information from a compacted RC-path, and therefore this last component of an RC-path ought to be stored in BFROM, alongside with the compacted path. This is possible, even if RC-paths are compacted before the computation of BFROM. Indeed, the first phase in the BFROM computation produces an intra procedural approximation to BFROM, in which any possible interprocedural link is stored as a link from an occurence back to a dummy occurence of the same variable in some call or entry instruction. This suffices to determine the last component of the RC-path of the final link.