Inter-procedural slicing using SDGs

K. V. Raghavan

IISc

System Dependence Graph (SDG)

- SDG an extension of PDG, to represent multi-procedure programs.
- An SDG consists of a collection of PDGs, one per procedure, with some additional nodes, and connected by special edges.
- Our primary reference: "Interprocedural slicing using dependence graphs" by S. Horwitz, T. Reps, and D. Binkley, 1990.

Assumptions

- Procedures end with **return** statements instead of **end** statements. A **return** statement does not include a list of variables.
- Parameters are passed by *value-result*.
  - That is, values of all parameters are copied from actual parameters to formal parameters during entry to a procedure, and in the reverse direction at the time of return. (For simplicity, assume each actual parameter is a variable, not an expression.)
  - Global variables can be handled as extra parameters; however, these are ignored for simplicity of presentation.
  - All other assumptions of intra-procedural slicing approach are still retained.

Additional nodes

(This part of the lecture is based on Section 3.1.)

- Consider any procedure *foo* (other than *main*), with *formal parameters* x and y.
- The CFG of *foo* is assumed to contain the following *formal-in* nodes right after the *entry* node: "$x = x\_in$" and "$y = y\_in$".
  - These extra variables are called *call temporary* variables.
- Also, the following *formal-out* nodes at the end: "$x\_out = x$" and "$y\_out = y$".
  - These extra variables are called *return temporary* variables.
- Procedures other than *main* don't have initial-definition or final-use nodes.
- *main* has initial-definition nodes and final-use nodes, as before.

Additional nodes - II

- Say *bar* calls *foo*.
- Say actual parameters passed by *bar* to *foo* are a, b.
- The following *actual-in* nodes are present just before the call-site: "$x\_in = a$" and "$y\_in = b$".
- The following *actual-out* nodes are present just after the call-site: "$a = x\_out$" and "$b = y\_out$".
- The call itself is represented as a *call-site* node "call *foo*".

Edges in the SDG

- Each PDG is constructed as usual from its CFG (with each CFG containing extra nodes, as mentioned earlier).
- Additionally:
  - A *call edge* (modeled as a control-dependence edge) is added from each call-site to the entry node of the called procedure.
  - *Parameter-in* edges are added from actual-in nodes at a call-site to the corresponding formal-in nodes in the called procedure.
  - *Parameter-out* edges are added from formal-out nodes of each procedure to corresponding actual-out nodes at each call-site. (Both parameter-in and parameter-out edges are modeled as flow-dependence edges.)
- See Fig. 4 in the paper for example SDG.

A naive approach

- Treat the entire SDG as if it were a single PDG. Do backwards transitive closure from criterion.
- This is sound. But is *context insensitive*, and hence imprecise.
- For e.g., "*i := y_out*" in *main* in Fig. 4 becomes dependent on "*x_in := sum*" in *main* (see Page 39).
  - Due to path: "*x_in := sum*" $\rightarrow$ "*x := x_in*" $\rightarrow$ "*a_in := x*" $\rightarrow$ "*a := a_in*" $\rightarrow$ "*a := a + b*" $\rightarrow$ "*a_out := a*" $\rightarrow$ "*z := a_out*" $\rightarrow$ "*z_out := z*" $\rightarrow$ "*y := z_out*" $\rightarrow$ "*y_out := y*" $\rightarrow$ "*i := y_out*"
- Solution:
  - Add *summary edges* from actual-ins to actual-outs at call-sites, so that during the graph traversal there is no need to *descend* into a called procedure (which would force us to *ascend* back into valid as well as invalid call sites).

Adding "summary" edges to the SDG

(This slide is a simplified version of the material in Sec. 3.2.)

do {
  if (in some procedure *bar* a formal-out node *w*
    is transitively dependent on a formal-in node *v* due to
    intra-procedural flow- and control-dependences in *bar*'s PDG)
  then
    at each call-site to *bar* add a *summary edge*
    (modeled as a flow-dependence edge) to the actual-out node
    corresponding to *w* from the actual-in node corresponding
    to *v*.
} until (no more summary edges can be added)

Adding "summary" edges to the SDG

(This slide is a simplified version of the material in Sec. 3.2.)

do {
  if (in some procedure *bar* a formal-out node *w*
    is transitively dependent on a formal-in node *v* due to
    intra-procedural flow- and control-dependences in *bar*'s PDG)
  then
    at each call-site to *bar* add a *summary edge*
    (modeled as a flow-dependence edge) to the actual-out node
    corresponding to *w* from the actual-in node corresponding
    to *v*.
} until (no more summary edges can be added)

See complete example SDG (including summary edges) in Fig. 8.

Slicing

Slicing is to be done *after* all summary edges have been added.

1. Phase 1 – the "ascending" phase.
   1. Mark the nodes in the criterion S.
   2. Do backward SDG traversal, ignoring parameter-out and def-order edges. Mark visited nodes.
2. Phase 2 – the "descending" phase.
   1. Starting from nodes marked in Phase 1, do backward SDG traversal, this time ignoring call, paramater-in, and def-order edges. Mark visited nodes.
3. All marked nodes belong to the slice.

(See Fig. 9 in the paper.)

Minor imprecision in the slice

Consider Fig. 8 in the paper. This is the example SDG with summary edges added.

- A slice back from the z_{out} formal-out node in *Inc* reaches z_{in} actual-in node in A. This in turn depends on the actual-out node "$y = b_{out}$".
- This causes the following four nodes to be pulled into the slice: the b_{out} formal-out node in *Add*, the b_{in} formal-in node in the same procedure, the corresponding the actual-in node in A, and finally the y_{in} formal-in node in A.
- However, it would have been enough to pull in only the y_{in} formal-in node into the slice (because of the z_{in} actual-in node in A that needs to be in the slice). The other three nodes represent imprecision.

Minor imprecision in slice - II

- The reason for the imprecision is that even though *Add* does not modify its parameter b, its b_{out} formal-out node gets pulled into the slice.
- In general, if a procedure gets pulled into the slice this way, no other node in the procedure except the corresponding formal-in node will get pulled in. Therefore, the imprecision is rather minor.
- In contrast, if we had used a naive context-insensitive backward traversal of the SDG (i.e., without summary edges, and in a single phase) then the entire call to *Add* in *A*, including both its actual-in nodes, will get pulled in. This will also cause the nodes that define *sum* in procedure *Main* to get pulled in. This is very imprecise.

Fix to improve precision

GMOD analysis:

- For each procedure, identify parameters that it does not modify. Since a procedure may modify its parameters indirectly by passing them to other procedures that end up modifying them, this identification itself needs an inter-procedural dataflow analysis.
- Such an analysis is called a *GMOD* (Global Modification) analysis. This is described in Section 4.2 of the paper.

Fix to improve precision

GMOD analysis:

- For each procedure, identify parameters that it does not modify. Since a procedure may modify its parameters indirectly by passing them to other procedures that end up modifying them, this identification itself needs an inter-procedural dataflow analysis.
- Such an analysis is called a *GMOD* (Global Modification) analysis. This is described in Section 4.2 of the paper.

After GMOD analysis is over, for each parameter x of a procedure p that p does not modify:

- Delete the x_{out} formal-out node from p's CFG. Also, delete the corresponding actual-out nodes from all other CFGs that contain call-sites to p.

Fix to improve precision

GMOD analysis:

- For each procedure, identify parameters that it does not modify. Since a procedure may modify its parameters indirectly by passing them to other procedures that end up modifying them, this identification itself needs an inter-procedural dataflow analysis.
- Such an analysis is called a *GMOD* (Global Modification) analysis. This is described in Section 4.2 of the paper.

After GMOD analysis is over, for each parameter x of a procedure p that p does not modify:

- Delete the x_{out} formal-out node from p's CFG. Also, delete the corresponding actual-out nodes from all other CFGs that contain call-sites to p.

Then, rebuild all the PDGs, and the SDG, and do slicing as usual.

Executable slices

- During slicing it is possible that when a call-site gets pulled into the slice only a subset of its actual-in nodes get pulled into the slice.
  - For e.g., in the example in Fig. 8, with the z_{out} formal-out node in *Inc* as criterion, in the call-site in *Main* only the y_{in} actual-in gets included in the slice.
- Therefore, it is not straightforward to generate a proper sliced program from the set of SDG nodes that get marked during traversal.
- One solution to this: After slicing is over, put dummy constants in place of missing actual-in nodes at all call-sites.
  - These dummy constants will not affect the runtime behavior of the sliced program, because these actual-ins are anyway irrelevant to the criterion.
- Another solution is to generate different versions of a procedure for different subsets of actual-ins that are present at different call-sites to the procedure.

(See discussion in Page 28, Section 1 of the paper.)