Simplified Interprocedural Analysis Algorithm for Non-Recursive Programs

17-355/17-665/17-819: Program Analysis (Spring 2019)
Jonathan Aldrich (aldrich@cs.cmu.edu)

This simplified context-sensitive interprocedural analysis algorithm is capable of analyzing programs that do not contain recursion.

type $Context$
  val $fn : Function$ ▷ the function being called
  val $input : \sigma$ ▷ input for this set of calls

type $Summary$ ▷ the input/output summary for a context
  val $input : \sigma$
  val $output : \sigma$

val $results : Map[Context, Summary]$ ▷ the analysis results

function ANALYZE(ctx, σ_i)
  $\sigma'_o \leftarrow$INTRAPROCEDURAL(ctx, σ_i)
  $results[ctx] \leftarrow Summary(\sigma_i, results[ctx].output \sqcup \sigma'_o)$
  return σ'_o
end function

function FLOW($[\![n\text{: } x := f(y)]\!], ctx, \sigma_i$) ▷ called by intraprocedural analysis
  $\sigma_{in} \leftarrow [formal(f) \mapsto \sigma_i(y)]$ ▷ map f's formal parameter to info on actual from σ_i
  $calleeCtx \leftarrow$ GETCTX(f, ctx, n, σ_{in})
  $\sigma_o \leftarrow$RESULTSFOR($calleeCtx, \sigma_{in}$)
  return $\sigma_i[x \mapsto \sigma_o[result]]$ ▷ update dataflow with the function's result
end function

function RESULTSFOR(ctx, σ_i)
  $\sigma \leftarrow results[ctx].output$
  if $\sigma \neq \bot \wedge \sigma_i \sqsubseteq results[ctx].input$ **then**
    return σ ▷ existing results are good
  end if
  $results[ctx].input \leftarrow results[ctx].input \sqcup \sigma_i$ ▷ keep track of possibly more general input
  return ANALYZE($ctx, results[ctx].input$)
end function

function GETCTX($f, callingCtx, n, \sigma_i$)
  return $Context(f, \sigma_i)$ ▷ constructs a new $Context$ with f and σ_i
end function