

# CS738: Advanced Compiler Optimizations

## Interprocedural Data Flow Analysis

Amey Karkare

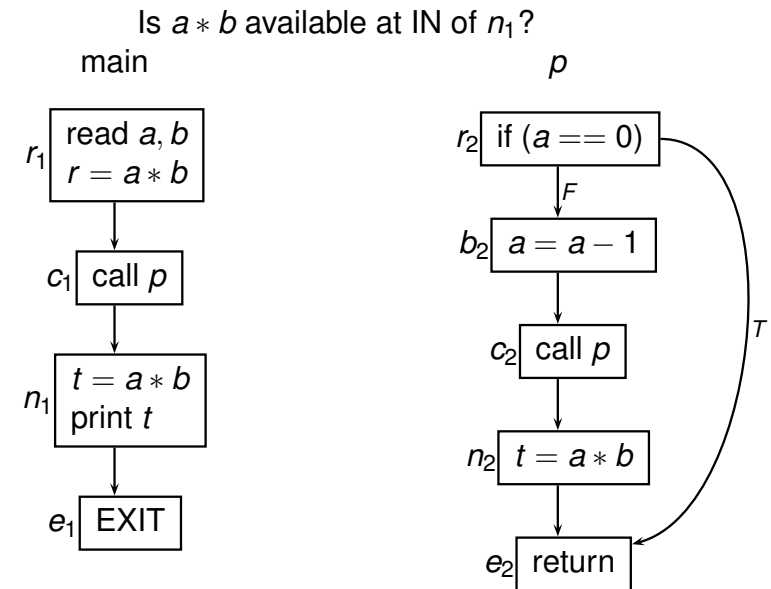
karkare@cse.iitk.ac.in

<http://www.cse.iitk.ac.in/~karkare/cs738>

Department of CSE, IIT Kanpur



## Interprocedural Analysis: WHY?

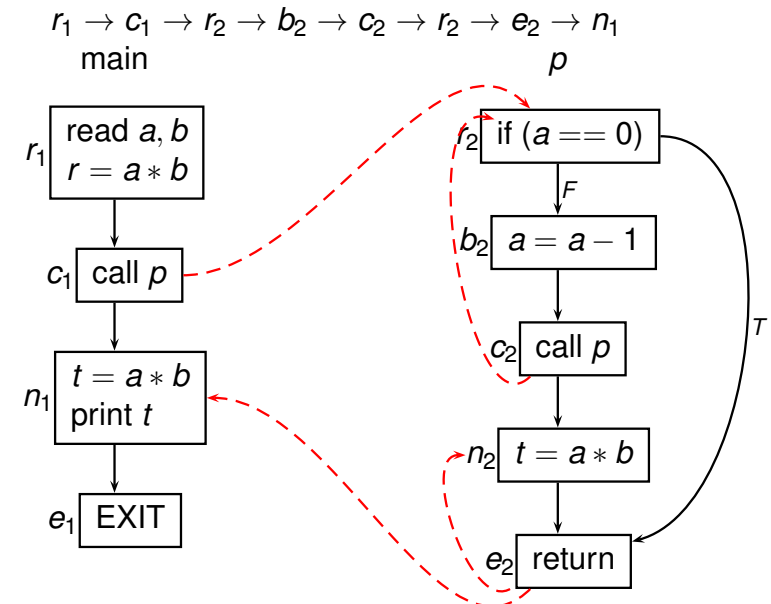


## Challenges

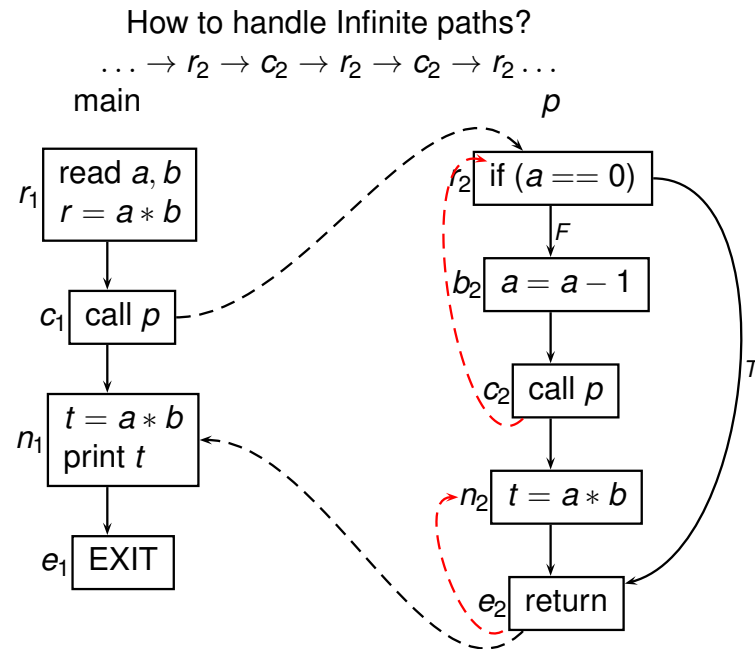
- Infeasible paths
- Recursion
- Function pointers and virtual functions
- Dynamic functions (functional programs)

## Infeasible Paths

How to avoid data flowing along invalid paths?



## Recursion



## Function Variables

- ▶ Target of a function can not be determined statically
- ▶ Function Pointers (including virtual functions)

```
double (*fun)(double arg);
...
if (cond)
    fun = sqrt;
else
    fun = fabs;
...
fun(x);
```
- ▶ Dynamically created functions (in functional languages)
- ▶ No static control flow graph!

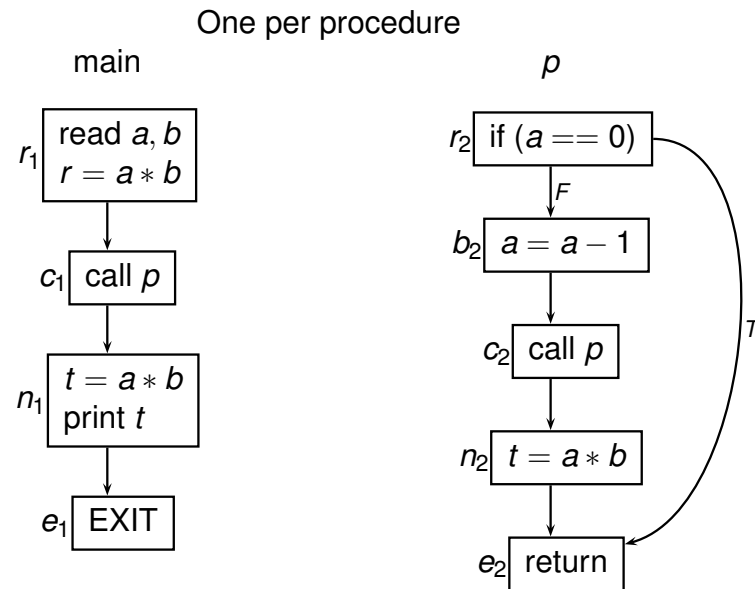
## Two Approaches

- ▶ Functional approach
  - ▶ procedures as structured blocks
  - ▶ input-output relation (*functions*) for each block
  - ▶ *function* used at call site to compute the effect of procedure on program state
- ▶ Call-strings approach
  - ▶ single flow graph for whole program
  - ▶ value of interest tagged with the history of unfinished procedure calls

## Notations and Terminology

M. Sharir, and A. Pnueli. **Two Approaches to Inter-Procedural Data-Flow Analysis**.  
In Jones and Muchnik, editors, Program Flow Analysis: Theory and Applications.  
Prentice-Hall, 1981.

## Control Flow Graph



## Control Flow Graph for Procedure $p$

- ▶ Single instruction basic blocks
- ▶ Unique exit block, denoted  $e_p$
- ▶ Unique entry block, denoted  $r_p$  (root block)
- ▶ Edge  $(m, n)$  if direct control transfer from (the end of) block  $m$  to (the start of) block  $n$
- ▶ Path:  $(n_1, n_2, \dots, n_k)$ 
  - ▶  $(n_i, n_{i+1}) \in \text{Edge set for } 1 \leq i < k$
  - ▶  $\text{path}_G(m, n)$ : Set of all path in graph  $G = (N, E)$  leading from  $m$  to  $n$

## Assumptions

- ▶ Parameterless procedures, to ignore the problems of
  - ▶ *aliasing*
  - ▶ recursion stack for formal parameters
- ▶ No procedure variables (pointers, virtual functions etc.)

## Data Flow Framework

- ▶  $(L, F)$ : data flow framework
- ▶  $L$ : a meet-semilattice
  - ▶ Largest element  $\Omega$
- ▶  $F$ : space of propagation functions
  - ▶ Closed under composition and meet
  - ▶ Contains  $id_L(x) = x$  and  $f_\Omega(x) = \Omega$
- ▶  $f_{(m,n)} \in F$  represents propagation function for edge  $(m, n)$  of control flow graph  $G = (N, E)$ 
  - ▶ Change of DF values from the *start* of  $m$ , through  $m$ , to the *start* of  $n$

## Data Flow Equations

$$\begin{aligned}x_r &= \textit{BoundaryInfo} \\x_n &= \bigwedge_{(m,n) \in E} f_{(m,n)}(x_m) \quad n \in N - r\end{aligned}$$

- MFP solution, approximation of MOP

$$y_n = \bigwedge \{f_p(\textit{BoundaryInfo}) : p \in \text{path}_G(r, n)\} \quad n \in N$$

## Functional Approach to Interprocedural Analysis

## Functional Approach

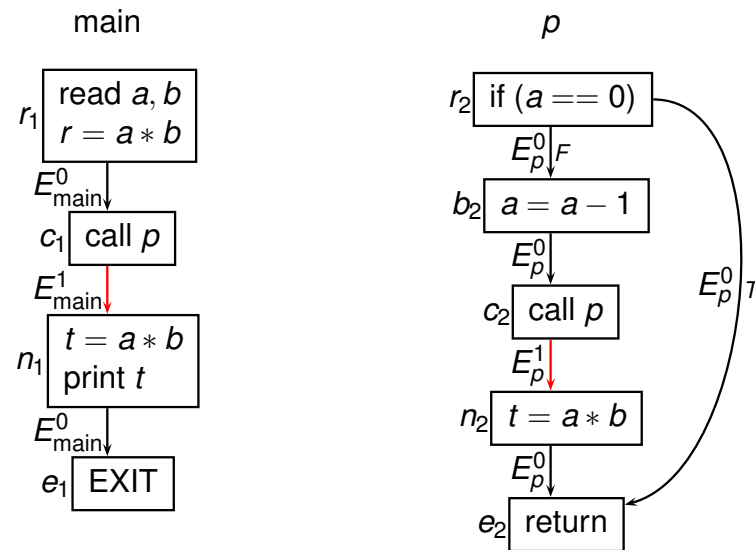
- Procedures treated as structures of blocks
- Computes relationship between DF value at entry node and related data at *any* internal node of procedure
- At call site, DF value propagated directly using the computed relation

## Interprocedural Flow Graph

First Representation:

$$\begin{aligned}G &= \bigcup \{G_p : p \text{ is a procedure in program}\} \\G_p &= (N_p, E_p, r_p) \\N_p &= \text{set of all basic block of } p \\r_p &= \text{root block of } p \\E_p &= \text{set of edges of } p \\&= E_p^0 \cup E_p^1 \\(m, n) \in E_p^0 &\Leftrightarrow \text{direct control transfer from } m \text{ to } n \\(m, n) \in E_p^1 &\Leftrightarrow m \text{ is a call block, and } n \text{ immediately follows } m\end{aligned}$$

## Interprocedural Flow Graph: 1<sup>st</sup> Representation



## Interprocedural Flow Graph

Second representation

$$G^* = (N^*, E^*, r_1)$$

$$r_1 = \text{root block of main}$$

$$N^* = \bigcup_p N_p$$

$$E^* = E^0 \cup E^1$$

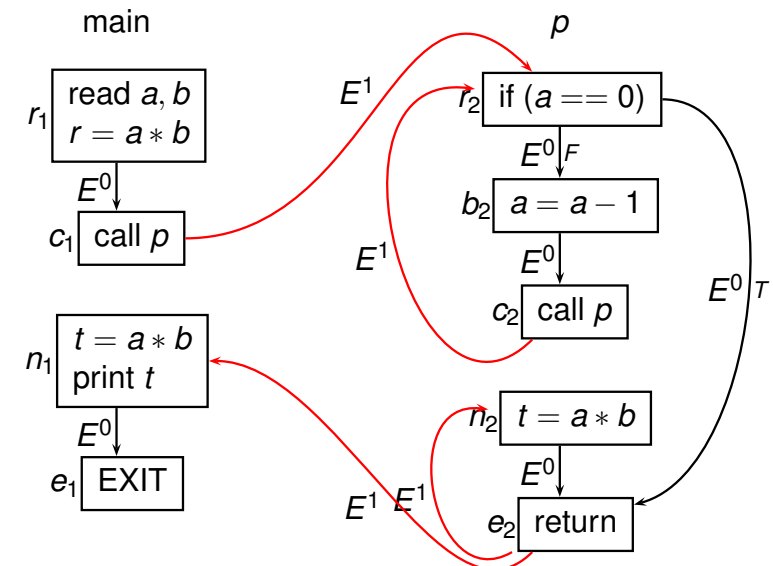
$$E^0 = \bigcup_p E_p^0$$

$$(m, n) \in E^1 \Leftrightarrow (m, n) \text{ is either a call edge or a return edge}$$

## Interprocedural Flow Graph

- Call edge  $(m, n)$ :
  - $m$  is a call block, say calling  $p$
  - $n$  is root block of  $p$
- Return edge  $(m, n)$ :
  - $m$  is an exit block of  $p$
  - $n$  is a block immediately following a call to  $p$
- Call edge  $(m, r_p)$  corresponds to return edge  $(e_q, n)$ 
  - if  $p = q$  and
  - $(m, n) \in E_s^1$  for some procedure  $s$

## Interprocedural Flow Graph: 2<sup>nd</sup> Representation



## Interprocedurally Valid Paths

- ▶  $G^*$  ignores the special nature of call and return edges
- ▶ Not all paths in  $G^*$  are feasible
  - ▶ do not represent potentially valid execution paths
- ▶  $IVP(r_1, n)$ : set of all interprocedurally valid paths from  $r_1$  to  $n$
- ▶ Path  $q \in \text{path}_{G^*}(r_1, n)$  is in  $IVP(r_1, n)$ 
  - ▶ iff sequence of all  $E^1$  edges in  $q$  (denoted  $q_1$ ) is *proper*

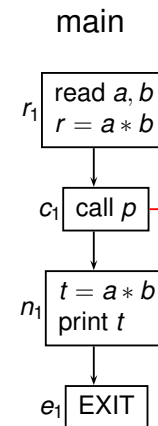
## Proper sequence

- ▶  $q_1$  without any return edge is proper
- ▶ let  $q_1[i]$  be the first return edge in  $q_1$ .  $q_1$  is proper if
  - ▶  $i > 1$ ; and
  - ▶  $q_1[i-1]$  is call edge corresponding to  $q_1[i]$ ; and
  - ▶  $q'_1$  obtained from deleting  $q_1[i-1]$  and  $q_1[i]$  from  $q_1$  is proper

## Interprocedurally Valid Complete Paths

- ▶  $IVP_0(r_p, n)$  for procedure  $p$  and node  $n \in N_p$
- ▶ set of all interprocedurally valid paths  $q$  in  $G^*$  from  $r_p$  to  $n$  s.t.
  - ▶ Each call edge has corresponding return edge in  $q$  restricted to  $E^1$

## IVPs



$r_1 \rightarrow c_1 \rightarrow r_2 \rightarrow c_2 \rightarrow r_2 \rightarrow e_2 \rightarrow n_2 \rightarrow e_2 \rightarrow n_1 \rightarrow e_1 \in IVP(r_1, e_1)$

## Path Decomposition

$$q \in \text{IVP}(r_{\text{main}}, n)$$

$\Leftrightarrow$

$$q = q_1 \parallel (c_1, r_{p_2}) \parallel q_2 \parallel \cdots \parallel (c_{j-1}, r_{p_j}) \parallel q_j$$

where for each  $i < j$ ,  $q_i \in \text{IVP}_0(r_{p_i}, c_i)$  and  $q_j \in \text{IVP}_0(r_{p_j}, n)$