

# **Data-flow Analysis: Lecture 6 & 7**

## **Interprocedural Analysis: Call-strings**

### **Technique [Sharir-Pnueli]**

Deepak D'Souza

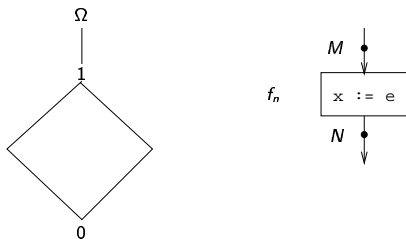
Department of Computer Science and Automation  
Indian Institute of Science, Bangalore.

October 31, 2007

## Sharir-Pnueli framework

- Continuation of Kildall's framework
- Finite lattice of data values
- Distributive data-flow functions
- Extend to handling programs with procedures

## Recap: Distributive data-flow framework



- Interested in computing “meet over all paths” (MOP) value:

$$x_N = \bigwedge_{\text{paths } \rho \text{ ending in } N} f_\rho(0).$$

- Setup data-flow equations:

$$x_N = f_n(x_M) \\ \dots$$

- so that MOP is maximum solution to these equations.

## Recap of Kildall's algorithm

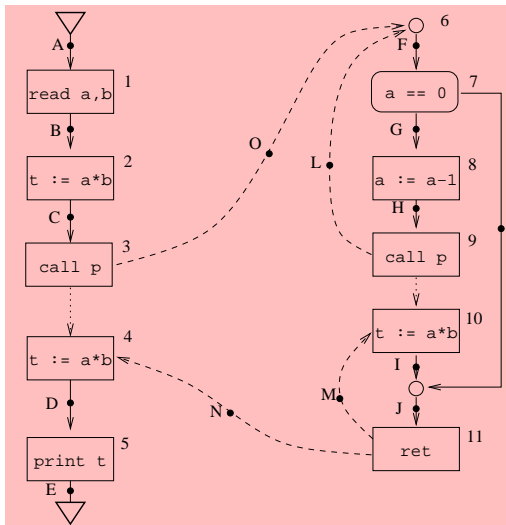
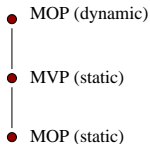
- Initialize value at each program point to  $\top$ .
- Mark initial nodes
- Choose a marked node  $n$ , unmark it, and propagate value to its successors based on  $f_n$ , by taking meet with existing value.
- Mark successor nodes for which new value is smaller than old value.
- Stop when no marked nodes are left.

### Fact

- Kildall's algo computes **maximum solution** to equations when data flow functions are **distributive**.
- Guaranteed to **converge** when lattice is **finite** or has **no infinite decreasing chains**.

## Extending to programs with procedure calls

- We want MOP values at each program point.
- Ignoring procedurally “valid” paths loses too much data.
- Can we compute “MVP” (meet over valid paths) values instead?

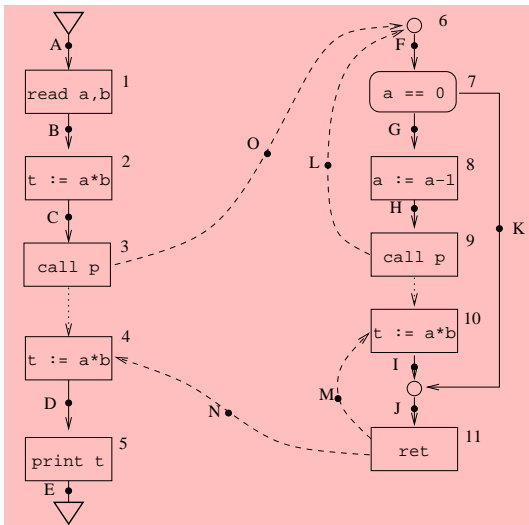


## Problems with naive approach

- Try to set up equations for  $x_N$  (MVP at program point  $N$ ).

- 

$$x_N = \bigwedge_{p \in IVP(r_1, N)} f_p(0).$$



## Functional Approach: Equations for $\phi_{r_p, n}$

$$\begin{aligned}\phi_{r_p, r_p} &\leq id_L \\ \phi_{r_p, n} &= \bigwedge_{(m, n) \in E_p} (h_{m, n} \cdot \phi_{r_p, m})\end{aligned}$$

where

$$h_{(m, n)} = \begin{cases} f_{m, n} & \text{if } (m, n) \in E_p^0 \\ \phi_{r_q, e_q} & \text{if } (m, n) \in E_p^1, \text{ and } m \text{ calls } q \end{cases}$$

## Equations for $x_n$

$$\begin{aligned}x_1 &= 0 \\x_{r_p} &= \bigwedge_{\text{calls } c \text{ to } p \text{ in } q} \phi_{r_q, c}(x_{r_q}) \\x_n &= \phi_{r_p, n}(x_{r_p}) \quad n \text{ in } N_p - \{r_p\}\end{aligned}$$

## Call-Strings approach

Basic idea:

- “Tag” data value for a path with call-string along that path
- Move over to lattice of vectors indexed by call-strings,  $L^*$ .
- Now set up data-flow equations to compute MVP values (indexed by each call-string).
- Finally “flatten” vectors to get MVP value

## Tagging with call-strings

- Call-string associated with path  $p$ :  $CM(p)$  = sequence of un-returned calls in  $p$ .
- Classify paths reaching  $N$  according to call-strings
- For each call-string  $\gamma$  maintain data value

$$d = \bigwedge_{p \in CM^{-1}(\gamma)} f_p(0).$$

- Thus elements of  $L^*$  are maps  $\xi : \Gamma \rightarrow D$ , and ordering  $\xi_1 \leq \xi_2$  is pointwise extension of  $\leq$  in  $L$ .
- Tagged MVP value:  $x_N^* : \gamma \mapsto \bigwedge_{p \in CM^{-1}(\gamma)} f_p(0)$ .
- MVP value  $x_n = \bigwedge_{\gamma \in \Gamma} x_n^*(\gamma)$ .

## Equations for tagged data values

- Data-flow functions for tagged values:

$$f_n^* : \xi \mapsto \xi'$$

- If  $n$  is not a call/ret node:

$$\xi'(\gamma) = f_n(\xi(\gamma)).$$

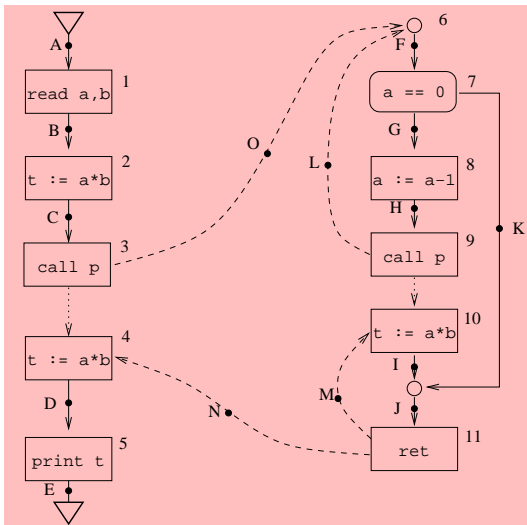
- If  $n$  is a call node  $c$ :

$$\xi'(\gamma \cdot c) = \xi(\gamma).$$

- If  $n$  is a return node  $e$  for call  $c$ :

$$\xi'(\gamma) = \xi(\gamma \cdot c).$$

- $\xi'(\gamma) = \Omega$  otherwise

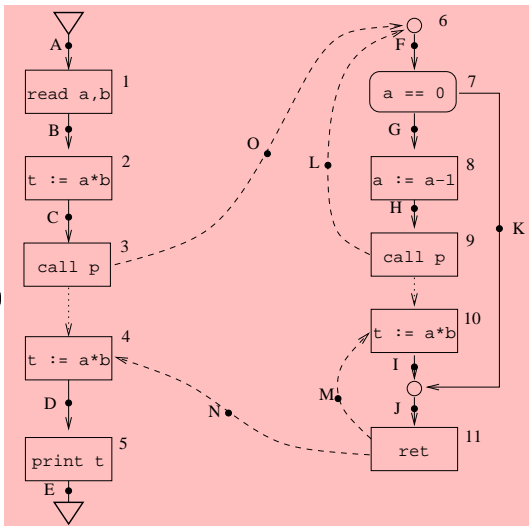


## Equations for tagged data values

$$\begin{aligned}x_n^* &\leq (\epsilon \mapsto 0) \\x_n^* &= \bigwedge_{(m,n) \in E^*} (f_m^*(x_m^*))\end{aligned}$$

# Example

$$\begin{aligned}
 x_{r_1}^* &\leq (\epsilon \mapsto 0) \\
 x_n^* &= \bigwedge_{(m,n) \in E^*} (f_m^*(x_m^*))
 \end{aligned}$$



## Convergence of iteration

- Lattice  $L^*$  is infinite for recursive programs.
- Possible to bound the size of call strings  $\Gamma$  we need to consider.

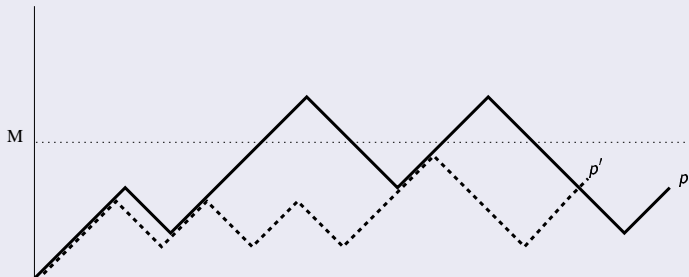
### Claim

For any path  $p$  with  $|CM(p)| > K(|L|)^2 = M$  there is a shorter path  $q$  with  $|CM(q)| \leq M$  and  $f_p(0) = f_q(0)$ .

## Ensuring convergence

- Go over to a finite lattice.
- Consider only call strings of length  $\leq M$  (Call this  $\Gamma_M$ ) for a certain  $M$ .
- $IVP_{\Gamma_M}(r_1, N) =$  paths from  $r_1$  to  $N$  such that for each prefix  $q$ ,  $CM(q) \leq M$ .

### Paths with bounded call-strings



## Ensuring convergence – ctd

- Consider paths  $p \in IVP_{\Gamma_M}$ .
- Show that for each path  $p \in IVP(r_1, N)$  we have a path  $p' \in IVP_{\Gamma_M}$  with  $f_{p'}(0) = f_p(0)$ .
- Observe that

$$\text{MVP at } N = \bigwedge_{\gamma} x_N^*(\gamma) = \bigwedge_{\gamma \in \Gamma_M} x_N^*(\gamma) = \text{MVP}_{\Gamma_M} \text{ at } N.$$

- Set up data-flow equations to capture  $\text{MVP}_{\Gamma_M}$

## Data-flow functions to capture $MVP_{\Gamma_M}$

- Data-flow functions for  $\Gamma_M$  tagged values:  $f_n^* : \xi \mapsto \xi'$ .
- If  $n$  is not a call/ret node:

$$\xi'(\gamma) = f_n(\xi(\gamma)).$$

- If  $n$  is a call node  $c$ :

$$\xi'(\gamma \cdot c) = \xi(\gamma) \text{ if } \gamma \cdot c \in \Gamma_M.$$

- If  $n$  is a return node  $e$  for call  $c$ :

$$\xi'(\gamma) = \xi(\gamma \cdot c) \text{ if } \gamma \cdot c \in \Gamma_M..$$

- $\xi'(\gamma) = \Omega$  otherwise (undefined).

## Bounding call-string size of paths

### Claim

For any path  $p$  in  $IVP(r_1, N)$  with  $|CM(p)| > M = K|L|^2$  there is a path  $p'$  in  $IVP_{\Gamma_M}(r_1, N)$  with  $f_{p'}(0) = f_p(0)$ .

- Sufficient to prove:

### Subclaim

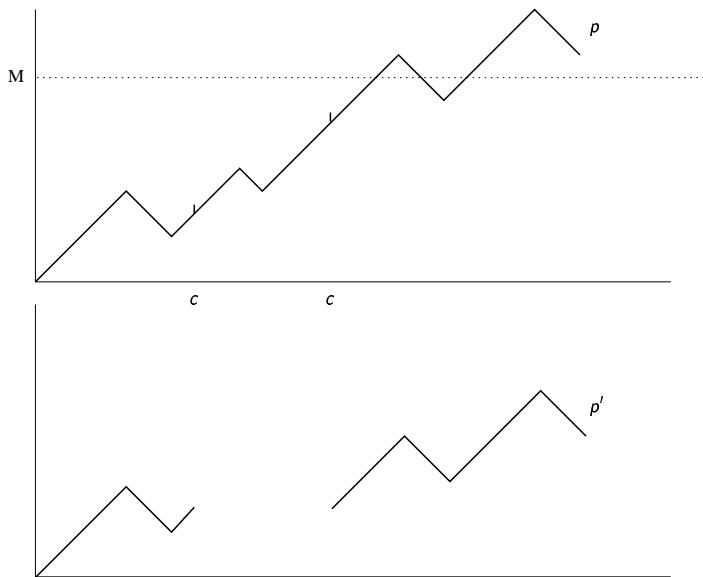
For any path  $p$  in  $IVP(r_1, N)$  with a prefix  $q$  such that  $|CM(q)| > M$ , we can produce a **smaller** path  $p'$  in  $IVP(r_1, N)$  with  $f_{p'}(0) = f_p(0)$ .

- ...since if  $|p| \leq M$  then  $p \in IVP_{\Gamma_M}$ .

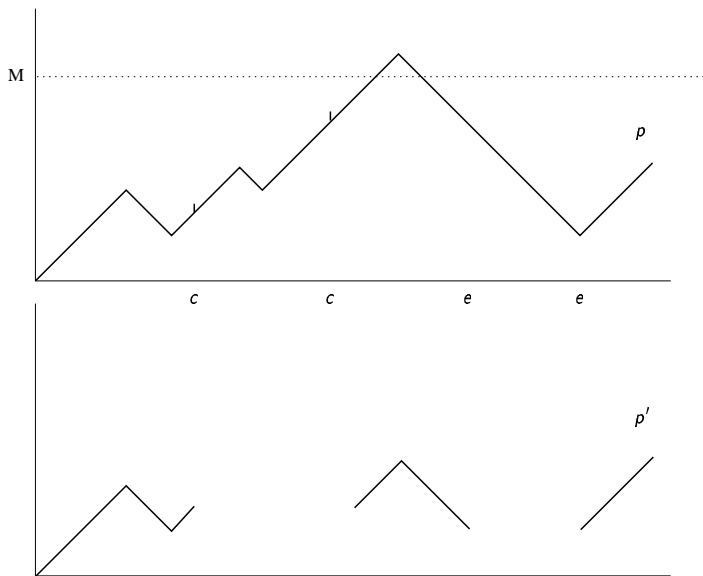
## Proving subclaim

- Let  $p_0$  be the first prefix of  $p$  where  $|CM| > M$ .
- Tag each unfinished-call prefix  $q \cdot c$  in  $p_0$  by  $(c, f_{q \cdot c}(0), f_{q \cdot cq'e})$  where  $e$  is corresponding return of  $c$  in  $p$ .
- If no return for  $c$  in  $p$  tag with  $(c, f_{q \cdot c}(0), \Omega)$ .
- Number of distinct such tags is  $K \cdot |L|$ , where  $K$  is number of calls nodes in the program.
- So there are two calls  $qc$  and  $qcq'c$  with same tag values.

## Proving subclaim – tag values are $\Omega$



## Proving subclaim – tag values are not $\Omega$



# Example

