SPARSE INTERPROCEDURAL DATAFLOW ANALYSIS

Mingshan March 29

Data Flow Analysis Basics

Precise Interprecedural Dataflow Analysis

Sparse Precise Interprecedural Dataflow Analysis

1.DATA FLOW ANALYSIS BASICS

Compiler

- Perform analysis based on local information

Data Flow Analysis

- Perform a more global analysis

- Discover more properties of program by associating an appropriate set of dataflow facts with each program point.

DATA FLOW ANALYSIS EXAMPLES

- Constant propagation (must, forward)
- Available expressions analysis (must, forward)
- Reaching definitions analysis (may, forward)
- Uninitialized variables analysis (may, forward)
- Live variables analysis (may, backward)
- Very busy expressions analysis (must, backward)
- .....

LIVE VARIABLES ANALYSIS

- A variable is live at a program point if its current value **may be read** in the **remaining execution**.

```
var x,y,z;
x = input;
while (x>1) {
    y = x/2;
    if (y>3)
        x = x-y;
    z = x-4;
    if (z>0)
        x = x/2;
    z = z-1;
}
output x;
```

Which variables are live at which locations?

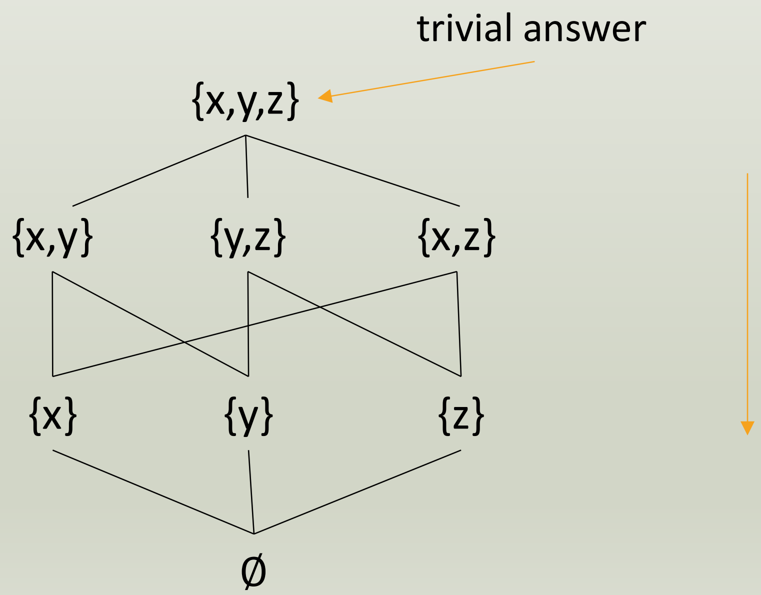
Which are not?

→ find out the set of live variables for each point.

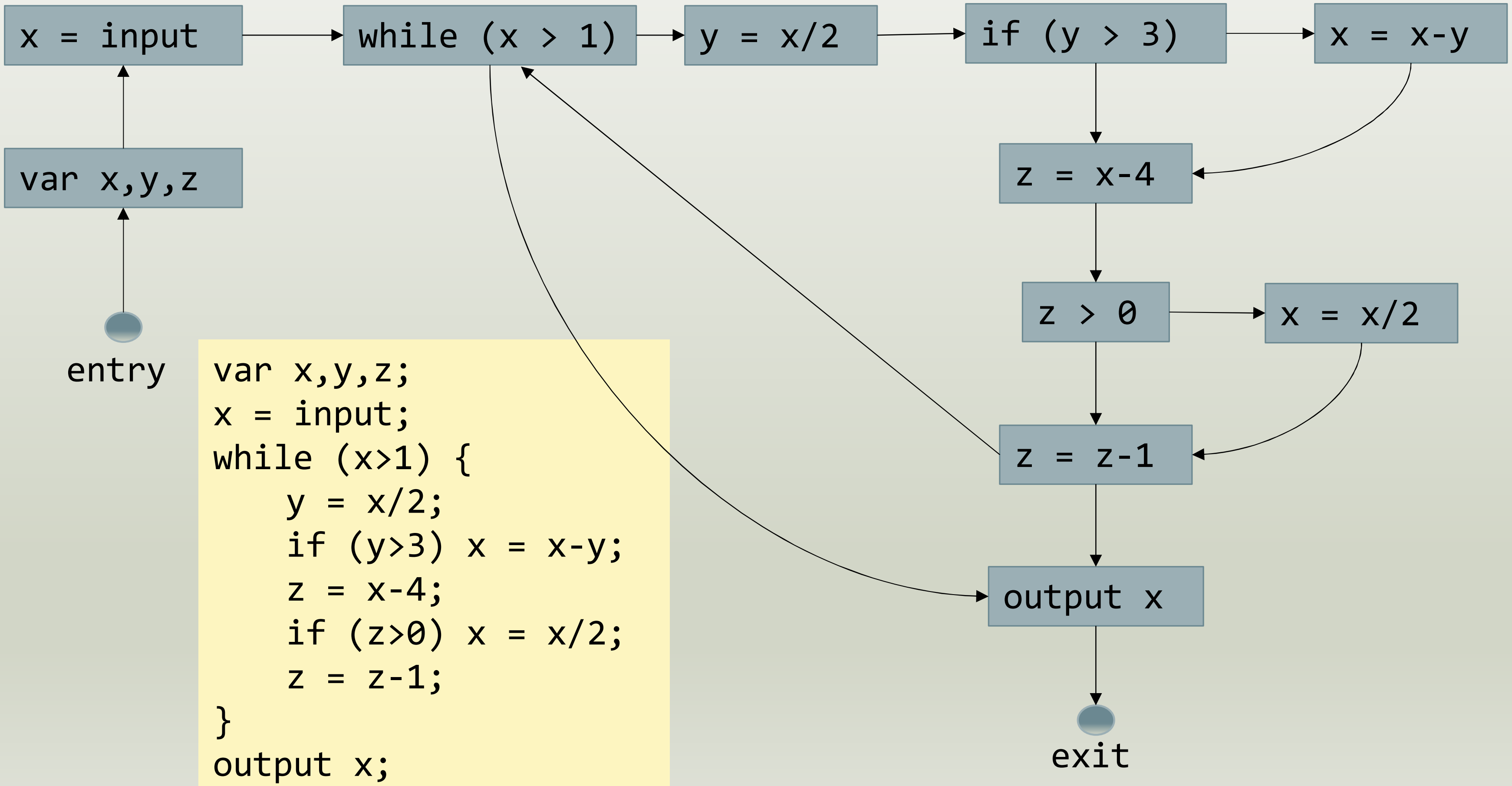
LATTICE

```
var x,y,z;
x = input;
while (x>1) {
    y = x/2;
    if (y>3) x = x-y;
    z = x-4;
    if (z>0) x = x/2;
    z = z-1;
}
output x;
```

$L = (2^{\{x,y,z\}}, \subseteq)$



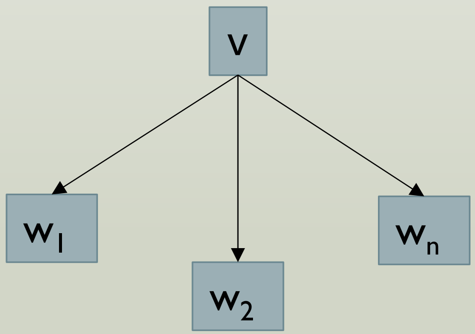
CONTROL FLOW GRAPH




```
var x,y,z;
x = input;
while (x>1) {
    y = x/2;
    if (y>3) x = x-y;
    z = x-4;
    if (z>0) x = x/2;
    z = z-1;
}
output x;
```

SETTING UP

- Let ⟦v⟧ denote the set of variables live before node v (pre-state of v)
- $Join(v) = \bigcup_{w \in succ(v)} [\![w]\!]$



CONSTRAINTS

- v is exit node:
  - $[\![exit]\!] = Join(v) \cup \emptyset = \emptyset$
- v is condition:
  - $[\![if\ (E)]\!] = [\![while(E)]\!] = Join(v) \cup vars(E)$
- v is output:
  - $[\![output\ E]\!] = join(v) \cup vars(E)$
- v is assignment:
  - $[\![x = E]\!] = (join(v) \setminus \{x\}) \cup vars(E)$
- v is variable declaration:
  - $[\![var\ x_1, \ldots, x_n]\!] = join(v) \setminus \{x_1, \ldots, x_n\}$
- v is other node:
  - $[\![v]\!] = join(v)$

var(E) = variables occurring (being read) in E

right hand side of each equation is monotonic

COMPUTING (LEAST) FIXED POINT

- $[\![\text{exit}]\!] = \emptyset$
- $[\![\text{output x}]\!] = [\![\text{exit}]\!] \cup \{x\}$ $= \{x\}$
- $[\![z = z - 1]\!] = ([\![\text{output x}]\!] \cup [\![\text{ while}(x > 1)]\!] \setminus \{z\}) \cup \{z\}$ $= \{x, z\}$
- $[\![x = x/2]\!] = ([\![z = z - 1]\!] \setminus \{x\}) \cup \{x\}$ $= \{x, z\}$
- $[\![\text{if}(z > 0)]\!] = [\![z = z - 1]\!] \cup [\![x = x/2]\!] \cup \{z\}$ $= \{x, z\}$
- $[\![z = x - 4]\!] = ([\![\text{if}(z > 0)]\!] \setminus \{z\}) \cup \{x\}$ $= \{x\}$
- $[\![x = x - y]\!] = ([\![z = x - 4]\!] \setminus \{x\}) \cup \{x, y\}$ $= \{x, y\}$
- $[\![\text{if}(y > 3)]\!] = [\![z = x - 4]\!] \cup [\![x = x - y]\!] \cup \{y\}$ $= \{x, y\}$
- $[\![y = x/2]\!] = ([\![\text{if}(y > 3)]\!] \setminus \{y\}) \cup \{x\}$ $= \{x\}$
- $[\![\text{while}(x > 1)]\!] = [\![\text{output x}]\!] \cup [\![y = x/2]\!] \cup \{x\}$ $= \{x\}$
- $[\![x = \text{input}]\!] = [\![\text{while}(x > 1)]\!] \setminus \{x\}$ $= \emptyset$
- $[\![\text{var x, y, z}]\!] = [\![x = \text{input}]\!] \setminus \{x, y, z\}$ $= \emptyset$
- $[\![\text{entry}]\!] = [\![\text{var x, y, z}]\!]$ $= \emptyset$

$[\![\text{while}(x > 1)]\!] = \{x, y, z\}$ //initialized to T

```
var x,y,z;
x = input;
while (x>1) {
    y = x/2;
    if (y>3) x = x-y;
    z = x-4;
    if (z>0) x = x/2;
    z = z-1;
}
output x;
```

Many non-trivial answer!

OPTIMIZATION BASED ON ANALYSIS

- y and z are never simultaneously live
  - they can share the same variable location
- z = z - 1 is never read
  - the assignment can be skipped

```
var x,y,z;
x = input;
while (x>1) {
    y = x/2;
    if (y>3) x = x-y;
    z = x-4;
    if (z>0) x = x/2;
    z = z-1;
}
output x;
```

```
var x,yz;
x = input;
while (x>1) {
    yz = x/2;
    if (yz>3) x = x-yz;
    yz = x-4;
    if (yz>0) x = x/2;
}
output x;
```

TWO KINDS OF PROBLEMS

Forward problems： the information at node N is based on the information of all its previous nodes.
- examples: Constant propagation, Reaching definitions

Backward problems： the information at node N is based on the information of all its successive nodes.
- examples: Very busy expressions analysis

MAY VS. MUST

- A may analysis (set union at join)
  - describes information that is possibly true
  - an over-approximation
  - examples: live variables, reaching definitions

- A must analysis (set intersection at join)
  - describes information that is definitely true
  - an under-approximation
  - examples: available expressions, very busy expressions

2. PRECISE INTERPROCEDURAL ANALYSIS VIA GRAPH REACHABILITY

To find *precise* solutions to *a large class of* interprocedural dataflow-analysis problems in polynomial time.

- 'precise' means providing 'meet-over-all-valid-paths' solution. (context-sensitive)
- 'a large class' consists of all problems in which the set of dataflow facts D is a ***finite** set* and the dataflow functions (which are in $2^D \rightarrow 2^D$) **distribute** over the confluence operator (either union or intersection, depending on the problem).

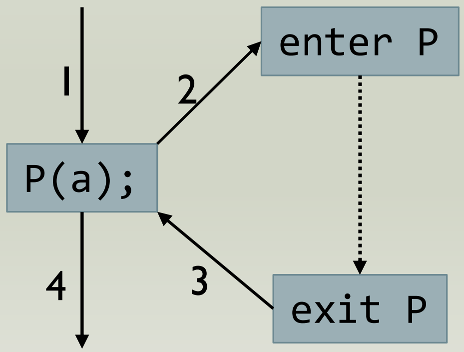
A function f: L → L is **distributive** iff for all x,y in L: f(x meet y) = f(x) meet f(y).

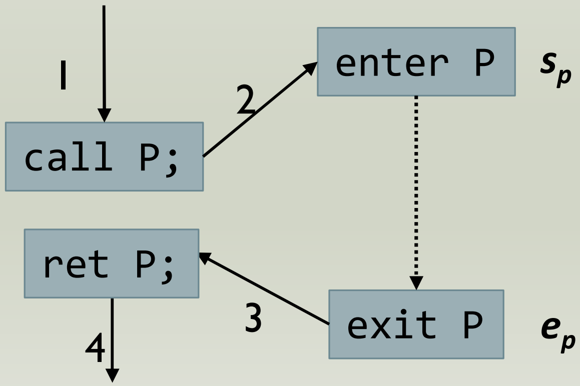
- → **IFDS problems** : interprocedural, finite, distributive, subset problems

IFDS FRAMEWORK

A program is represented using a directed graph $G^* = (N^*, E^*)$ called a ***supergraph***.

- G^* consists of a collection of control flow graphs $G1, G2, \ldots$ (one for each procedure), one of which, G_{main}, represents the program's main procedure.
- Each flowgraph G_p has a unique ***start* node s_p**, and a unique ***exit* node e_p**.
- A procedure call is represented by two nodes, a ***call* node** and a ***return-site* node**.
- Other nodes of the flowgraph represent the statements and predicates of the procedure in the usual way.



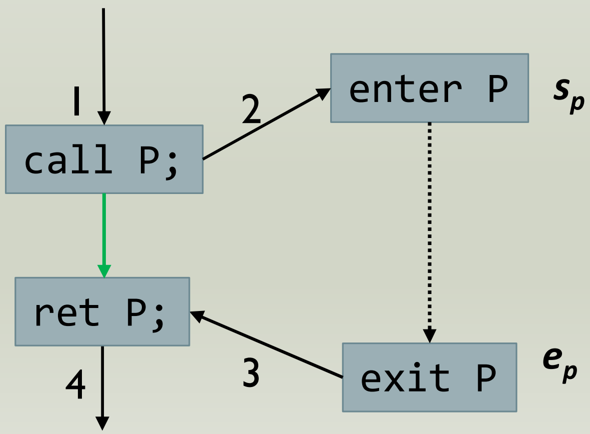


EDGES IN SUPERGRAPH

For each procedure call, represented by call-node *c* and return-site node *r*,

G * has three edges:

- An interprocedural ***call-to-start*** edge from *c* to the start node of the called procedure;
- An interprocedural ***exit-to-return-site*** edge from the exit node of the called procedure to *r*.
- An intraprocedural ***call-to-return-site*** edge from *c* to *r*; The call-to-return-site edges are included so that the IFDS framework can handle programs with local variables.



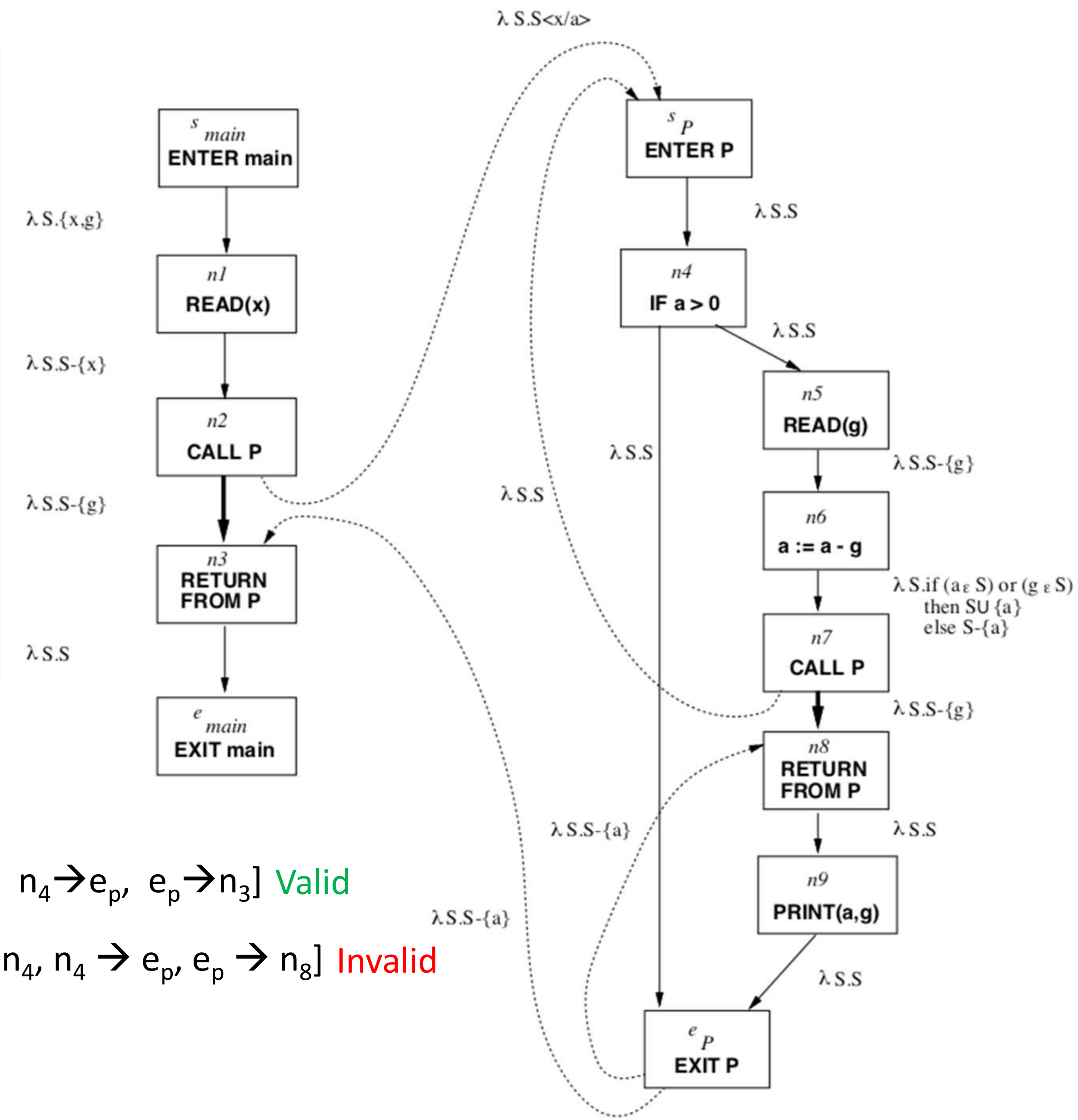
```
declare g: integer

program main
begin
  declare x: integ
  read(x)
  call P(x)
end

procedure P(va
begin
  if (a > 0) then
    read(g)
    a := a − g
    call P(a)
    print(a, g)
  fi
end
```

```
global var g;

void main(){
    var x;
    x = input;
    P(x);
}
void P(x){
    if (a > 0){
        g = input;
        a = a - g;
        P(a);
        print(a, g);
    }
}
```



[S_{main} → n_1, n_1 → n_2, n_2 → s_p, s_p → n_4, n_4 → e_p, e_p → n_3] Valid

[S_{main} → n_1, n_1 → n_2, n_2 → s_p, s_p → n_4, n_4 → e_p, e_p → n_8] Invalid

AN INSTANCE IP OF IFDS PROBLEM

$IP = (G\ , D, F, M, \lceil\ \rceil)$

I. G^* is a supergraph as defined above.

II. D is a finite set.

III. $F \subseteq 2^D \rightarrow 2^D$ is a set of distributive functions.

IV. M: $E^* \rightarrow F$ is a map from G^*'s edges to dataflow functions.

V. The meet operator is either union or intersection.

MVP SOLUTION TO IP

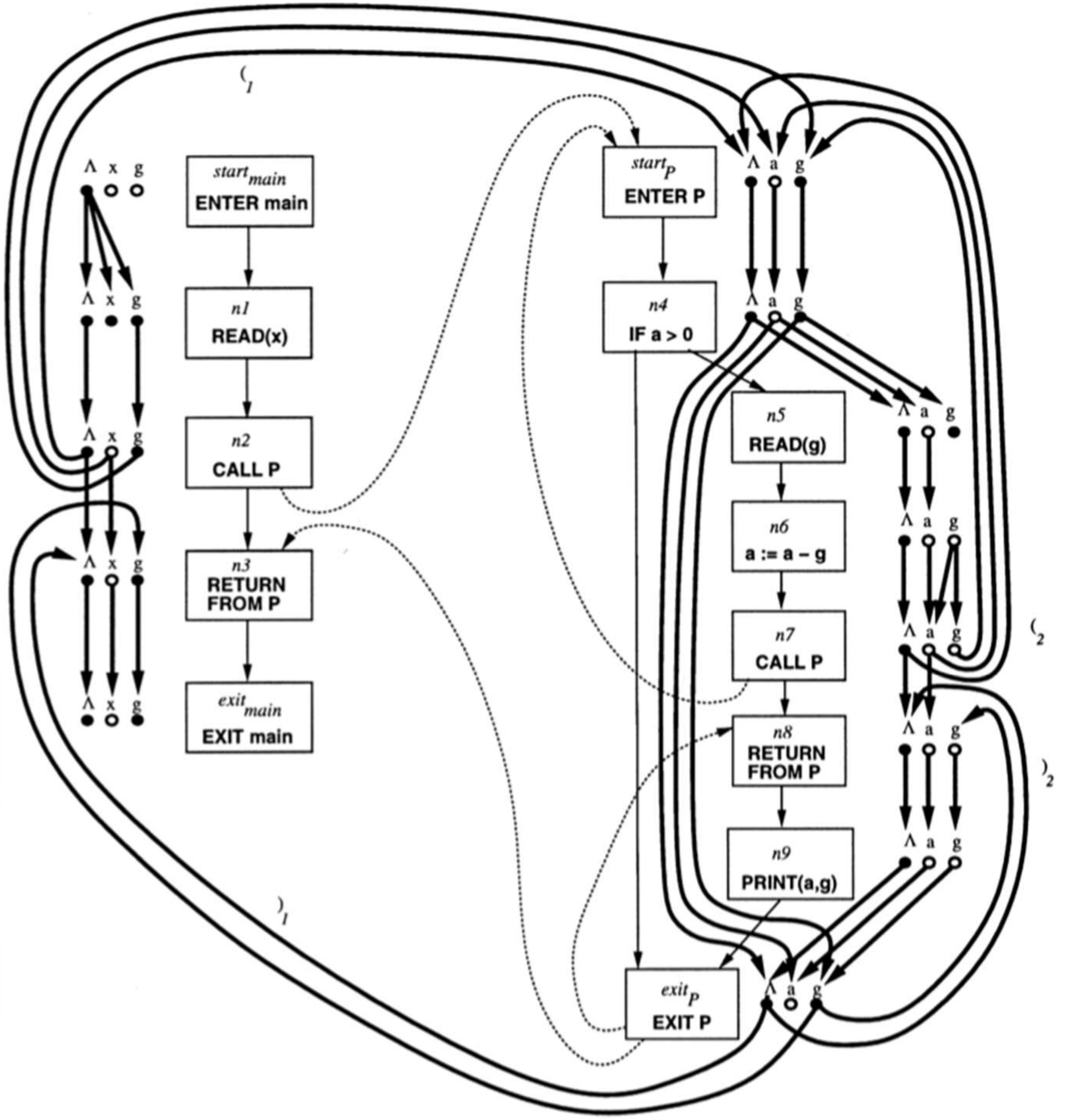
- Let *IP* = (*G**, *D*, *F*, *M*, ⌈ ⌉) be an IFDS problem instance. The ***meet-over-all-valid-paths*** solution to *IP* consists of the collection of values MVP_n defined as follows:

$$MVP_n = \bigsqcap_{q \in \mathrm{IVP}(s_{main},\, n)} pf_q(\top) \qquad \text{for each } n \in N^*$$

IFDS Problems

↓

Realizable-path
graph-reachability problems



TABULATION ALGORITHM

declare PathEdge, WorkList, SummaryEdge: **global** edge set

algorithm Tabulate($G^{\#}_{IP}$)
begin
  Let $(N^{\#}, E^{\#}) = G^{\#}_{IP}$
  PathEdge := $\{ \langle s_{main}, \mathbf{0} \rangle \rightarrow \langle s_{main}, \mathbf{0} \rangle \}$
  WorkList := $\{ \langle s_{main}, \mathbf{0} \rangle \rightarrow \langle s_{main}, \mathbf{0} \rangle \}$
  SummaryEdge := $\emptyset$
  ForwardTabulateSLRPs()
  for each $n \in N^{*}$ **do**
    $X_n := \{ d_2 \in D \mid \exists\, d_1 \in (D \cup \{ \mathbf{0} \})$ such that $\langle s_{procOf(n)}, d_1 \rangle \rightarrow \langle n, d_2 \rangle \in$ PathEdge $\}$
  od
end

procedure Propagate(e)
begin
  if $e \notin$ PathEdge **then** Insert e into PathEdge; Insert e into WorkList **fi**
end

procedure ForwardTabulateSLRPs()
begin
while WorkList $\neq \emptyset$ **do**
Select and remove an edge $\langle s_p, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ from WorkList
switch n

case $n \in Call_p$:
for each d_3 such that $\langle n, d_2 \rangle \rightarrow \langle s_{calledProc(n)}, d_3 \rangle \in E^{\#}$ **do**
Propagate($\langle s_{calledProc(n)}, d_3 \rangle \rightarrow \langle s_{calledProc(n)}, d_3 \rangle$)
od
for each d_3 such that $\langle n, d_2 \rangle \rightarrow \langle returnSite(n), d_3 \rangle \in (E^{\#} \cup$ SummaryEdge) **do**
Propagate($\langle s_p, d_1 \rangle \rightarrow \langle returnSite(n), d_3 \rangle$)
od
end case

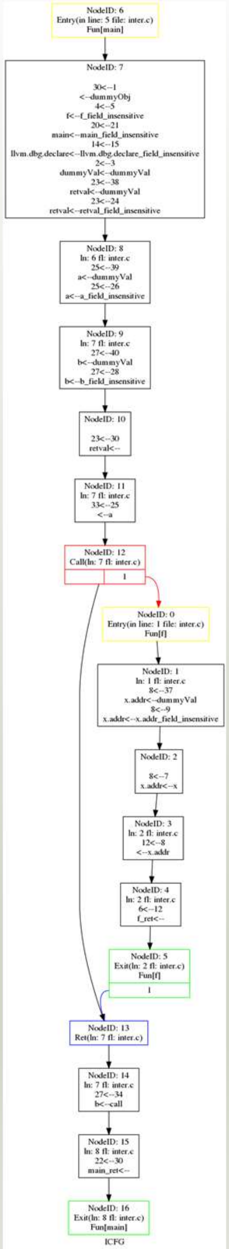
case $n = e_p$:
for each $c \in callers(p)$ **do**
for each d_4, d_5 such that $\langle c, d_4 \rangle \rightarrow \langle s_p, d_1 \rangle \in E^{\#}$ and $\langle e_p, d_2 \rangle \rightarrow \langle returnSite(c), d_5 \rangle \in E^{\#}$ **do**
if $\langle c, d_4 \rangle \rightarrow \langle returnSite(c), d_5 \rangle \notin$ SummaryEdge **then**
Insert $\langle c, d_4 \rangle \rightarrow \langle returnSite(c), d_5 \rangle$ into SummaryEdge
for each d_3 such that $\langle s_{procOf(c)}, d_3 \rangle \rightarrow \langle c, d_4 \rangle \in$ PathEdge **do**
Propagate($\langle s_{procOf(c)}, d_3 \rangle \rightarrow \langle returnSite(c), d_5 \rangle$)
od
fi
od
od
end case

case $n \in (N_p - Call_p - \{ e_p \})$:
for each $\langle m, d_3 \rangle$ such that $\langle n, d_2 \rangle \rightarrow \langle m, d_3 \rangle \in E^{\#}$ **do**
Propagate($\langle s_p, d_1 \rangle \rightarrow \langle m, d_3 \rangle$)
od
end case

end switch
od
end

```c
int f(int x){
	return x;
}

int main(){
	int a;
	int b = f(a);
	return 0;
}
```



```
Analysis Terminates! Possibly uninitialized variables are: {28 26}

ICFGNodeID:0: PAGNodeSet: {28 7 26}
ICFGNodeID:1: PAGNodeSet: {28 7 26}
ICFGNodeID:2: PAGNodeSet: {9 28 7 26}
ICFGNodeID:3: PAGNodeSet: {9 28 7 26}
ICFGNodeID:4: PAGNodeSet: {12 9 28 7 26}
ICFGNodeID:5: PAGNodeSet: {12 9 28 7 26 6}
ICFGNodeID:6: PAGNodeSet: {}
ICFGNodeID:7: PAGNodeSet: {}
ICFGNodeID:8: PAGNodeSet: {24}
ICFGNodeID:9: PAGNodeSet: {24 26}
ICFGNodeID:10: PAGNodeSet: {24 28 26}
ICFGNodeID:11: PAGNodeSet: {28 26}
ICFGNodeID:12: PAGNodeSet: {28 33 26}
ICFGNodeID:13: PAGNodeSet: {28 33 26 6}
ICFGNodeID:14: PAGNodeSet: {34 28 33 26 6}
ICFGNodeID:15: PAGNodeSet: {34 28 33 26 6}
ICFGNodeID:16: PAGNodeSet: {34 28 33 26 6}

*********** PathEdge **************
[ICFGNodeID:6,()] --> [ICFGNodeID:6,()]
[ICFGNodeID:6,()] --> [ICFGNodeID:7,()]
[ICFGNodeID:6,()] --> [ICFGNodeID:8,(24)]
[ICFGNodeID:6,()] --> [ICFGNodeID:9,(24 26)]
[ICFGNodeID:6,()] --> [ICFGNodeID:10,(24 28 26)]
[ICFGNodeID:6,()] --> [ICFGNodeID:11,(28 26)]
[ICFGNodeID:6,()] --> [ICFGNodeID:12,(28 33 26)]
[ICFGNodeID:6,()] --> [ICFGNodeID:13,(28 33 26)]
[ICFGNodeID:0,(28 7 26)] --> [ICFGNodeID:0,(28 7 26)]
[ICFGNodeID:0,(28 7 26)] --> [ICFGNodeID:1,(28 7 26)]
[ICFGNodeID:0,(28 7 26)] --> [ICFGNodeID:2,(9 28 7 26)]
[ICFGNodeID:0,(28 7 26)] --> [ICFGNodeID:3,(9 28 7 26)]
[ICFGNodeID:0,(28 7 26)] --> [ICFGNodeID:4,(12 9 28 7 26)]
[ICFGNodeID:0,(28 7 26)] --> [ICFGNodeID:5,(12 9 28 7 26 6)]
[ICFGNodeID:6,()] --> [ICFGNodeID:13,(6)]
[ICFGNodeID:6,()] --> [ICFGNodeID:14,(34 6)]
[ICFGNodeID:6,()] --> [ICFGNodeID:15,(34 28 6)]
[ICFGNodeID:6,()] --> [ICFGNodeID:16,(34 28 6)]
[ICFGNodeID:6,()] --> [ICFGNodeID:14,(28 33 26)]
[ICFGNodeID:6,()] --> [ICFGNodeID:15,(33 26)]
[ICFGNodeID:6,()] --> [ICFGNodeID:16,(33 26)]

*********** SummaryEdge **************
[ICFGNodeID:12,(28 33 26)] --> [ICFGNodeID:13,(6)]
-------------------------------------------------------
```

3. MAKE IT SPARSE

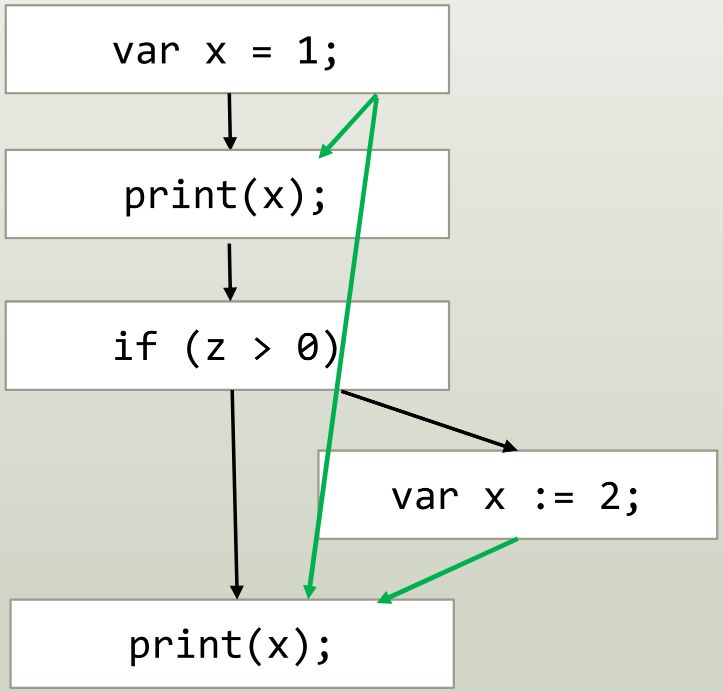
```
int main(){
        int a;
        int b;
        int c;
        int d;
        d = 3;
        c = 2;
        b = a + d;
        return 0;
}
```

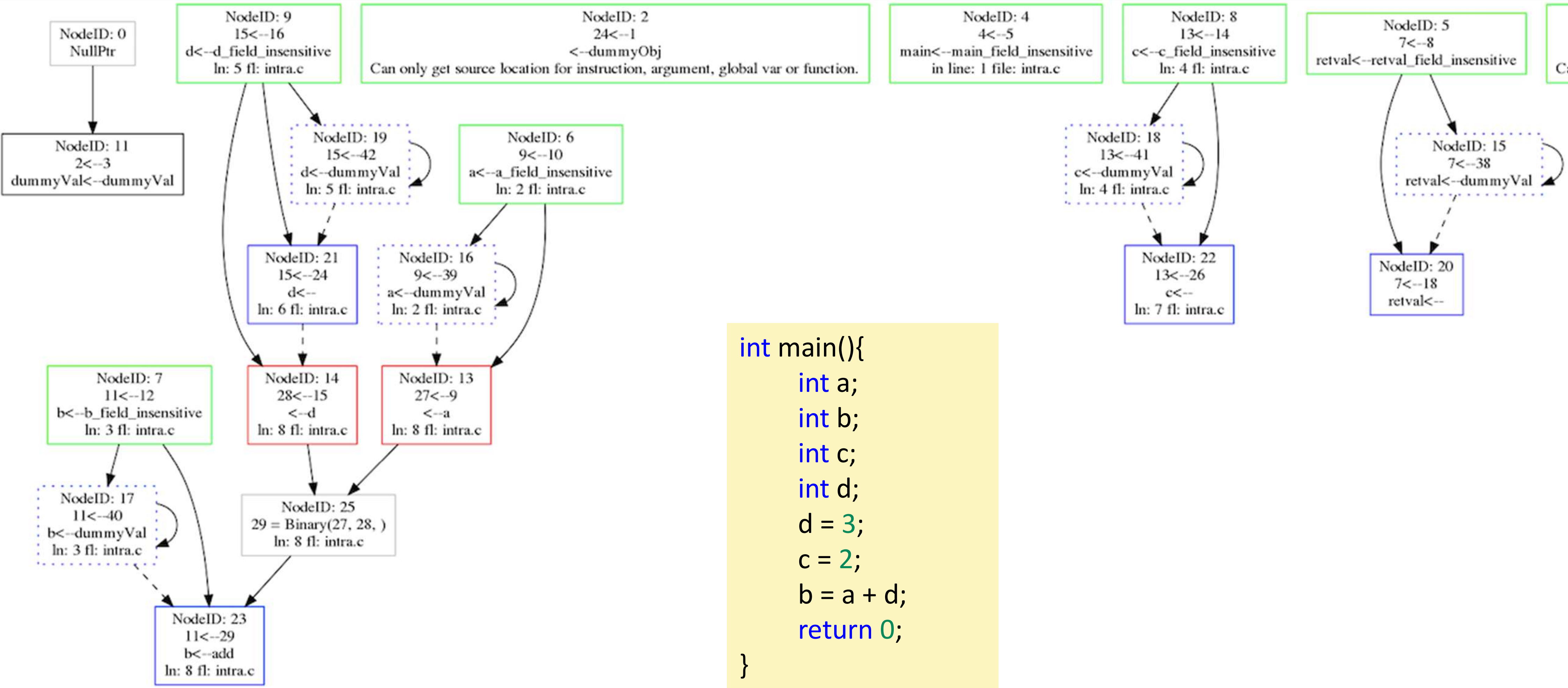
```
Analysis Terminates! Possibly uninitialized variables are: {12 10}

ICFGNodeID:0: PAGNodeSet: {}
ICFGNodeID:1: PAGNodeSet: {}
ICFGNodeID:2: PAGNodeSet: {8}
ICFGNodeID:3: PAGNodeSet: {10 8}
ICFGNodeID:4: PAGNodeSet: {12 10 8}
ICFGNodeID:5: PAGNodeSet: {12 10 8 14}
ICFGNodeID:6: PAGNodeSet: {12 10 8 16 14}
ICFGNodeID:7: PAGNodeSet: {12 10 16 14}
ICFGNodeID:8: PAGNodeSet: {12 10 14}
ICFGNodeID:9: PAGNodeSet: {12 10}
ICFGNodeID:10: PAGNodeSet: {27 12 10}
ICFGNodeID:11: PAGNodeSet: {27 12 10}
ICFGNodeID:12: PAGNodeSet: {29 27 12 10}
ICFGNodeID:13: PAGNodeSet: {29 27 12 10}
ICFGNodeID:14: PAGNodeSet: {29 27 12 10}

*********** PathEdge **************
[ICFGNodeID:0,()] --> [ICFGNodeID:0,()]
[ICFGNodeID:0,()] --> [ICFGNodeID:1,()]
[ICFGNodeID:0,()] --> [ICFGNodeID:2,(8)]
[ICFGNodeID:0,()] --> [ICFGNodeID:3,(10 8)]
[ICFGNodeID:0,()] --> [ICFGNodeID:4,(12 10 8)]
[ICFGNodeID:0,()] --> [ICFGNodeID:5,(12 10 8 14)]
[ICFGNodeID:0,()] --> [ICFGNodeID:6,(12 10 8 16 14)]
[ICFGNodeID:0,()] --> [ICFGNodeID:7,(12 10 16 14)]
[ICFGNodeID:0,()] --> [ICFGNodeID:8,(12 10 14)]
[ICFGNodeID:0,()] --> [ICFGNodeID:9,(12 10)]
[ICFGNodeID:0,()] --> [ICFGNodeID:10,(27 12 10)]
[ICFGNodeID:0,()] --> [ICFGNodeID:11,(27 12 10)]
[ICFGNodeID:0,()] --> [ICFGNodeID:12,(29 27 12 10)]
[ICFGNodeID:0,()] --> [ICFGNodeID:13,(29 27 12 10)]
[ICFGNodeID:0,()] --> [ICFGNodeID:14,(29 27 12 10)]
```

CONTROL FLOW VS. VALUE FLOW






```c
int main(){
    int a;
    int b;
    int c;
    int d;
    d = 3;
    c = 2;
    b = a + d;
    return 0;
}
```

```
ICFGNodeID:0: PAGNodeSet: {}
ICFGNodeID:1: PAGNodeSet: {}
ICFGNodeID:2: PAGNodeSet: {8}
ICFGNodeID:3: PAGNodeSet: {10 8}
ICFGNodeID:4: PAGNodeSet: {10 8 12}
ICFGNodeID:5: PAGNodeSet: {10 8 12 14}
ICFGNodeID:6: PAGNodeSet: {10 8 12 14 16}
ICFGNodeID:7: PAGNodeSet: {10 12 14 16}
ICFGNodeID:8: PAGNodeSet: {10 12 14}
ICFGNodeID:9: PAGNodeSet: {10 12}
ICFGNodeID:10: PAGNodeSet: {10 12 27}
ICFGNodeID:11: PAGNodeSet: {10 12 27}
ICFGNodeID:12: PAGNodeSet: {29 10 12 27}
ICFGNodeID:13: PAGNodeSet: {29 10 12 27}
ICFGNodeID:14: PAGNodeSet: {29 10 12 27}

*********** PathEdge **************
[ICFGNodeID:0,()] --> [ICFGNodeID:0,()]
[ICFGNodeID:0,()] --> [ICFGNodeID:1,()]
[ICFGNodeID:0,()] --> [ICFGNodeID:2,(8)]
[ICFGNodeID:0,()] --> [ICFGNodeID:3,(10 8)]
[ICFGNodeID:0,()] --> [ICFGNodeID:4,(10 8 12)]
[ICFGNodeID:0,()] --> [ICFGNodeID:5,(10 8 12 14)]
[ICFGNodeID:0,()] --> [ICFGNodeID:6,(10 8 12 14 16)]
[ICFGNodeID:0,()] --> [ICFGNodeID:7,(10 12 14 16)]
[ICFGNodeID:0,()] --> [ICFGNodeID:8,(10 12 14)]
[ICFGNodeID:0,()] --> [ICFGNodeID:9,(10 12)]
[ICFGNodeID:0,()] --> [ICFGNodeID:10,(10 12 27)]
[ICFGNodeID:0,()] --> [ICFGNodeID:11,(10 12 27)]
[ICFGNodeID:0,()] --> [ICFGNodeID:12,(29 10 12 27)]
[ICFGNodeID:0,()] --> [ICFGNodeID:13,(29 10 12 27)]
[ICFGNodeID:0,()] --> [ICFGNodeID:14,(29 10 12 27)]
```

Dataflow fact is propagated along CFG

Sparse Precise Interprocedural Dataflow Analysis

Dataflow fact is only propagated to where it will be used

```
SVFGNodeID:0: PAGNodeSet: {}
SVFGNodeID:5: PAGNodeSet: {}
SVFGNodeID:6: PAGNodeSet: {}
SVFGNodeID:7: PAGNodeSet: {}
SVFGNodeID:8: PAGNodeSet: {}
SVFGNodeID:9: PAGNodeSet: {}
SVFGNodeID:11: PAGNodeSet: {}
SVFGNodeID:13: PAGNodeSet: {<9,0> <10,1>}
SVFGNodeID:14: PAGNodeSet: {<15,0> <16,0>}
SVFGNodeID:15: PAGNodeSet: {<7,0>}
SVFGNodeID:16: PAGNodeSet: {<9,0>}
SVFGNodeID:17: PAGNodeSet: {<11,0>}
SVFGNodeID:18: PAGNodeSet: {<13,0>}
SVFGNodeID:19: PAGNodeSet: {<15,0>}
SVFGNodeID:20: PAGNodeSet: {<8,1> <7,0>}
SVFGNodeID:21: PAGNodeSet: {<15,0> <16,1>}
SVFGNodeID:22: PAGNodeSet: {<13,0> <14,1>}
SVFGNodeID:23: PAGNodeSet: {<11,0> <29,0> <29,1> <12,1>}
SVFGNodeID:25: PAGNodeSet: {<27,1> <28,0>}

*********** PathEdge **************
[SVFGNodeID:7|0,()] --> [SVFGNodeID:7,()]
[SVFGNodeID:9|0,()] --> [SVFGNodeID:9,()]
[SVFGNodeID:6|0,()] --> [SVFGNodeID:6,()]
[SVFGNodeID:8|0,()] --> [SVFGNodeID:8,()]
[SVFGNodeID:5|0,()] --> [SVFGNodeID:5,()]
[SVFGNodeID:5|0,()] --> [SVFGNodeID:15,(<7,0>)]
[SVFGNodeID:5|0,()] --> [SVFGNodeID:20,(<7,0>)]
[SVFGNodeID:5|0,()] --> [SVFGNodeID:20,(<8,1>)]
[SVFGNodeID:8|0,()] --> [SVFGNodeID:18,(<13,0>)]
[SVFGNodeID:8|0,()] --> [SVFGNodeID:22,(<13,0>)]
[SVFGNodeID:8|0,()] --> [SVFGNodeID:22,(<14,1>)]
[SVFGNodeID:6|0,()] --> [SVFGNodeID:13,(<9,0>)]
[SVFGNodeID:6|0,()] --> [SVFGNodeID:16,(<9,0>)]
[SVFGNodeID:6|0,()] --> [SVFGNodeID:13,(<10,1>)]
[SVFGNodeID:6|0,()] --> [SVFGNodeID:25,(<27,1>)]
[SVFGNodeID:6|0,()] --> [SVFGNodeID:23,(<29,1>)]
[SVFGNodeID:9|0,()] --> [SVFGNodeID:14,(<15,0>)]
[SVFGNodeID:9|0,()] --> [SVFGNodeID:19,(<15,0>)]
[SVFGNodeID:9|0,()] --> [SVFGNodeID:21,(<15,0>)]
[SVFGNodeID:9|0,()] --> [SVFGNodeID:14,(<16,0>)]
[SVFGNodeID:9|0,()] --> [SVFGNodeID:25,(<28,0>)]
[SVFGNodeID:9|0,()] --> [SVFGNodeID:23,(<29,0>)]
[SVFGNodeID:9|0,()] --> [SVFGNodeID:21,(<16,1>)]
[SVFGNodeID:7|0,()] --> [SVFGNodeID:17,(<11,0>)]
[SVFGNodeID:7|0,()] --> [SVFGNodeID:23,(<11,0>)]
[SVFGNodeID:7|0,()] --> [SVFGNodeID:23,(<12,1>)]
```