Precise Interprocedural Dataflow Analysis via Graph Reachability

Thomas Reps Susan Horwitz Mooly Sagiv

POPL 1995

Presenter: Ben Greenman

IFDS

Thomas Reps Susan Horwitz Mooly Sagiv

POPL 1995

Presenter: Ben Greenman

The IFDS "framework"

- a *model* for dataflow problems
- a *uniform solution* to these problems
- a polynomial-time *algorithm*

Reaching Definitions

What statements are affected by a given definition?

$$\frac{\begin{array}{c}\textit{there is a path}\\ \textit{from } \mathbf{x := e'} \textit{ to } \mathbf{e}\end{array} \qquad \begin{array}{c}\mathbf{x} \textit{ is not re-defined}\\ \textit{along the path}\end{array}}{\mathbf{x := e'} \textit{ reaches } \mathbf{e}}$$

Available Expressions

What expressions can a statement re-use?

$$\frac{\textit{there is a path from } \mathbf{x := \ldots\ e'\ \ldots;} \textit{ to } \mathbf{e} \qquad \textit{no variable in } \mathbf{e'} \textit{ is re-defined along the path}}{\mathbf{e'} \textit{ is available at } \mathbf{e}}$$

Live Variable Analysis

What variables are referenced **at/after** a statement?

$$\frac{\begin{array}{c}\textit{there is a path}\\ \textit{from } \mathbf{e} \textit{ to } \mathbf{e'}\end{array} \qquad \mathbf{v} \textit{ is referenced at } \mathbf{e'}}{\mathbf{v} \textit{ is live at } \mathbf{e}}$$

Possibly-Uninitialized Variables

Which variables may be **null** at a given statement?

$$\frac{\begin{array}{c}\mathbf{x := e';}\\ \text{does not appear}\\ \text{on any path to } \mathbf{e}\end{array}}{\mathbf{x}\text{ may be }\mathbf{null}\text{ at }\mathbf{e}}$$

$$\frac{\begin{array}{c}\mathbf{x := e';}\\ \text{is the most recent binding}\\ \\ \exists\ \mathbf{y} \in \mathbf{e'}\ .\\ \mathbf{y}\text{ may be }\mathbf{null}\text{ at }\mathbf{e'}\end{array}}{\mathbf{x}\text{ may be }\mathbf{null}\text{ at }\mathbf{e}}$$

Program Slicing

"The algorithm described in this paper yields an **improved interprocedural-slicing algorithm** ...

6x as fast as the Horwitz-Reps-Binkley algorithm."

Speeding up Slicing
Reps, Horwitz, Sagiv, Rosay; FSE '94

PLDI 1988

Interprocedural Slicing Using Dependence Graphs
Susan Horwitz, Thomas Reps, David Binkley

The Program Summary Graph and Flow-Sensitive Interprocedural Data-Flow Analysis
David Callahan

Interprocedural Side-Effect Analysis in Linear Time
Keith D. Cooper, Ken Kennedy

Possibly-Uninitialized Variables

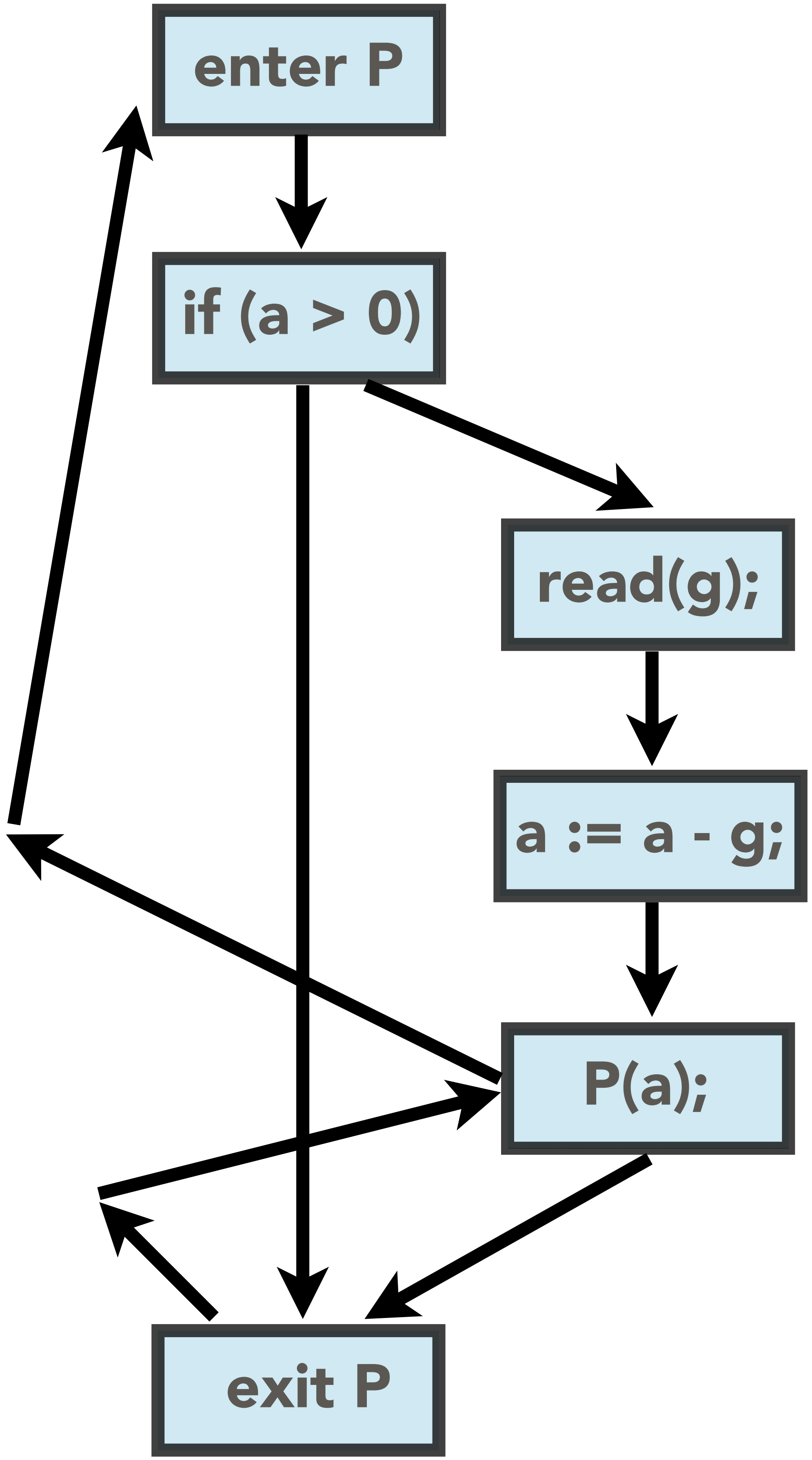
Which variables may be **null** at a given statement?

$$\frac{\begin{array}{c}\textbf{x := e';}\\ \text{does not appear}\\ \text{on any path to } \textbf{e}\end{array}}{x \text{ may be } \textbf{null} \text{ at e}}$$

$$\frac{\begin{array}{c}\textbf{x := e';}\\ \text{is the most recent binding}\\ \\ \exists\ \textbf{y} \in \textbf{e'}\ .\\ \textbf{y} \text{ may be } \textbf{null} \text{ at } \textbf{e'}\end{array}}{\textbf{x} \text{ may be } \textbf{null} \text{ at } \textbf{e}}$$

```
int g;

void P(int a) {
    if (a > 0) {
        read(g);
        a := a - g;
        P(a);
    }
}
```



enter P

if (a > 0)

read(g);

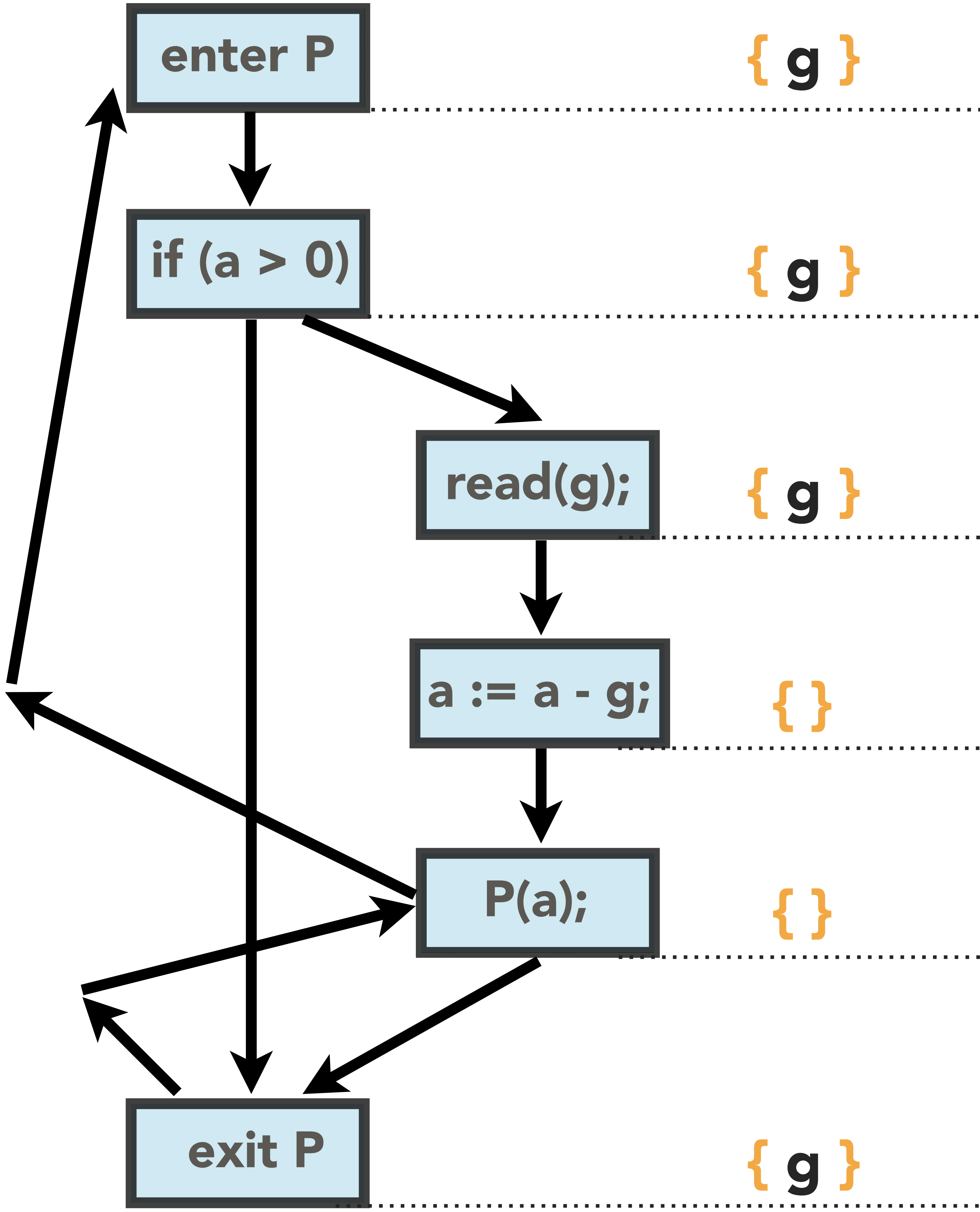
a := a - g;

P(a);

exit P

```
int g;

void P(int a) {
    if (a > 0) {
        read(g);
        a := a - g;
        P(a);
    }
}
```



enter P { g }

if (a > 0) { g }

read(g); { g }

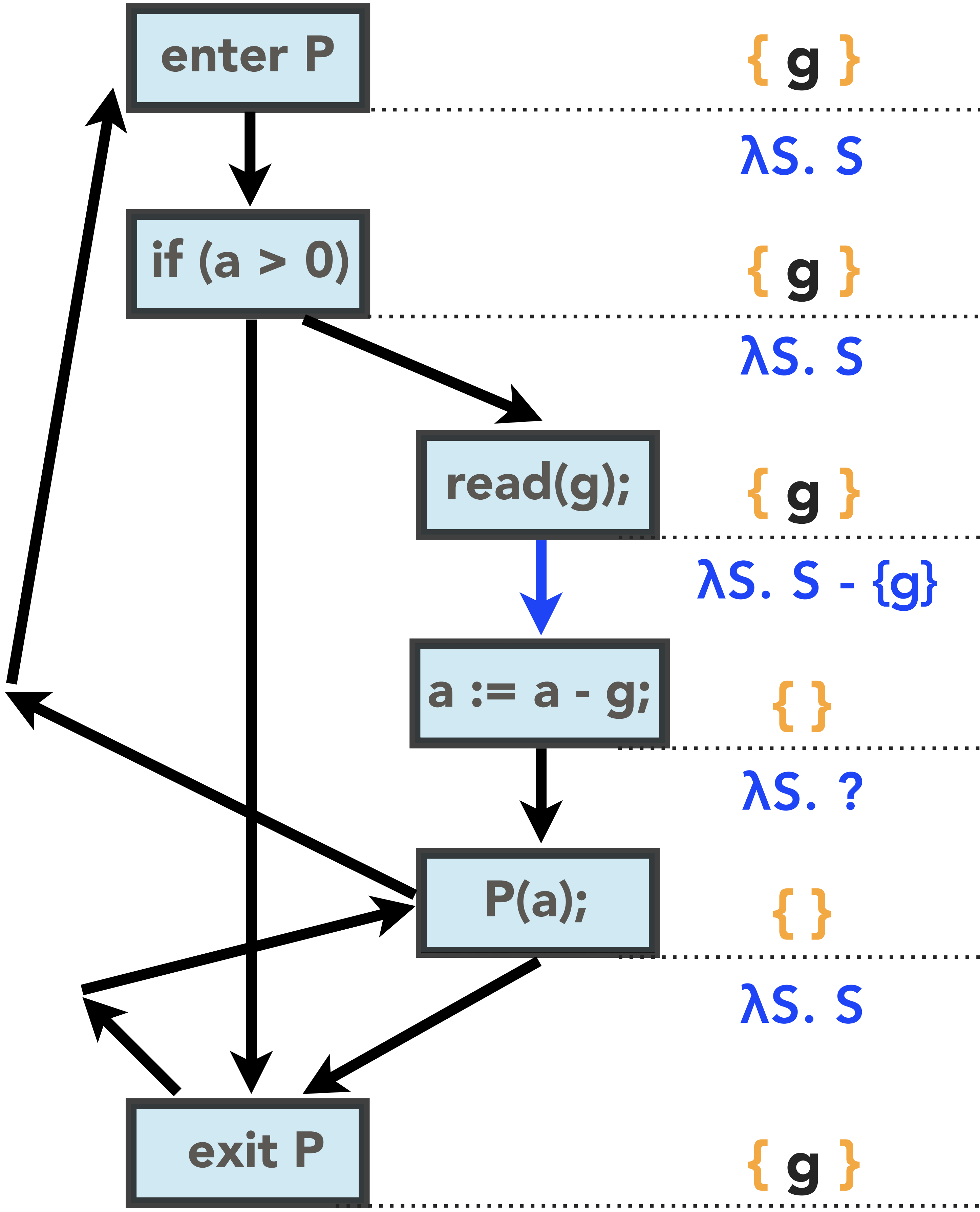
a := a - g; { }

P(a); { }

exit P { g }

```
int g;

void P(int a) {
    if (a > 0) {
        read(g);
        a := a - g;
        P(a);
    }
}
```



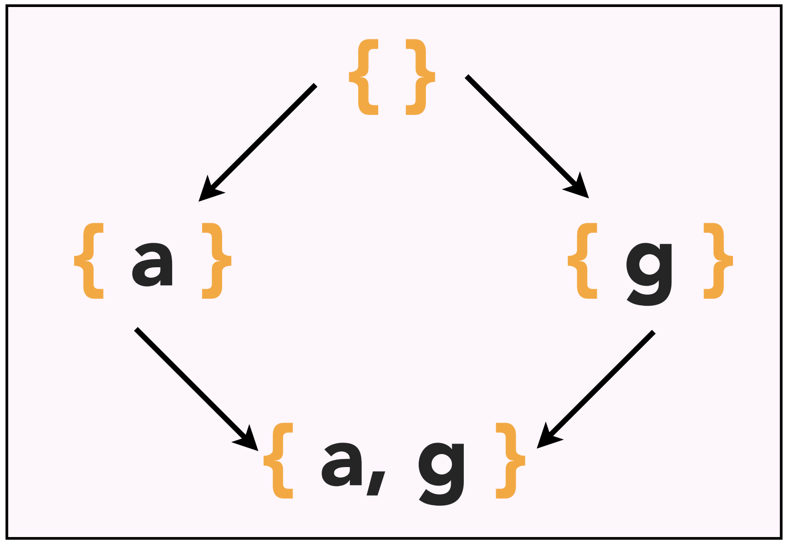
Kildall, POPL 1973

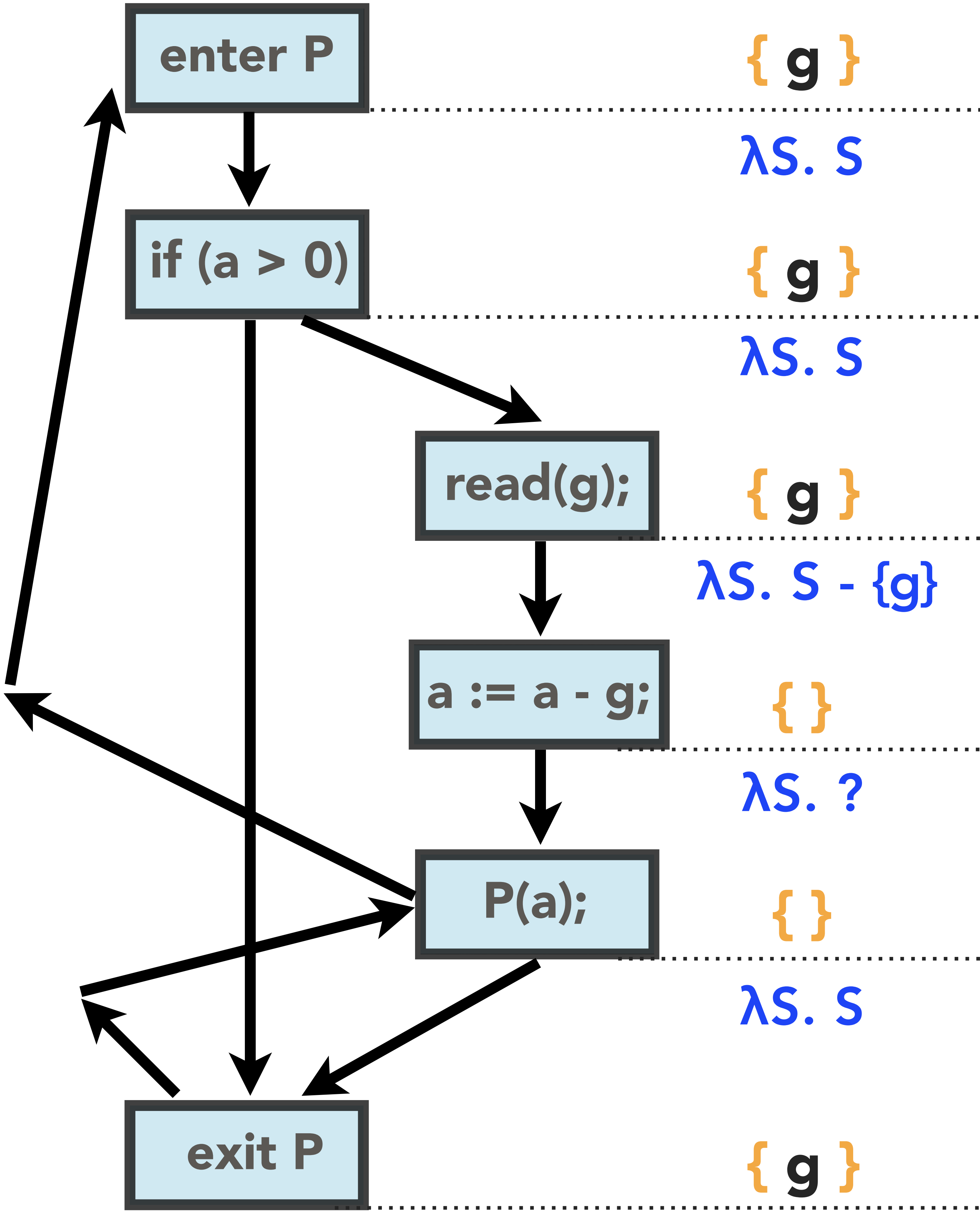
"Meet over all paths"

$$\bigsqcap f_n (\ldots (f_1 (\{ g \})) \ldots)$$

$f_1 \ldots f_n \in$ AllPaths

Infinite Set!

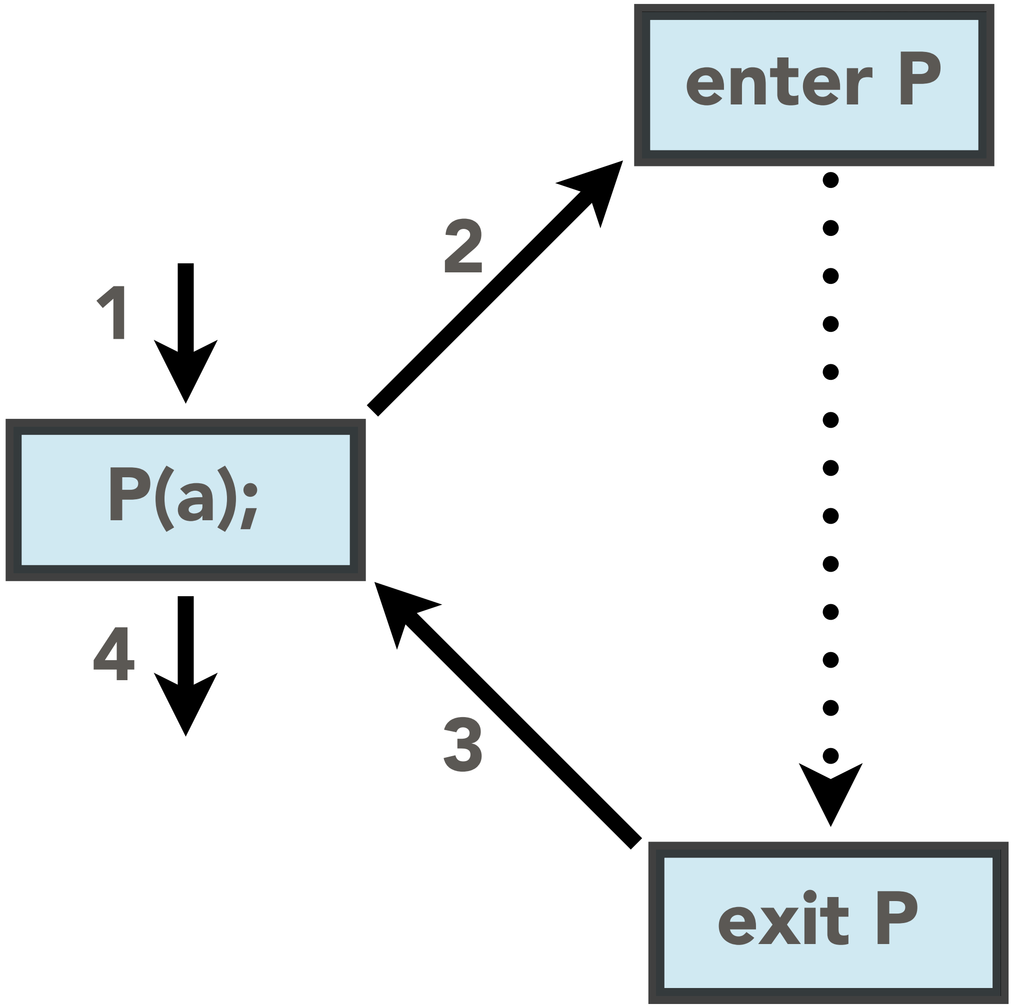




IFDS, POPL 1995

Meet over all **valid** paths

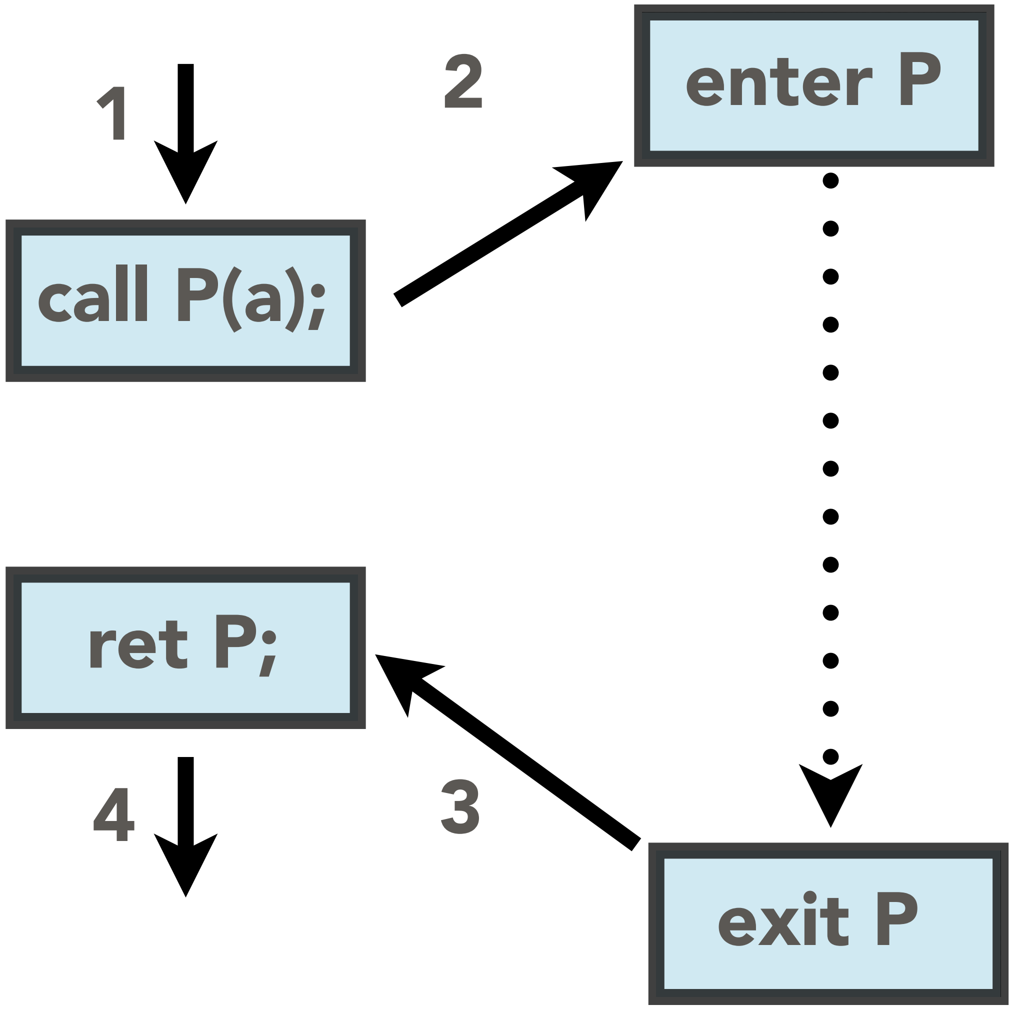
- Calls & Returns must match



IFDS, POPL 1995

Meet over all **valid** paths

- Calls & Returns must match
- Enforced by **call** & **ret** nodes



1

2

3

4

call P(a);

enter P

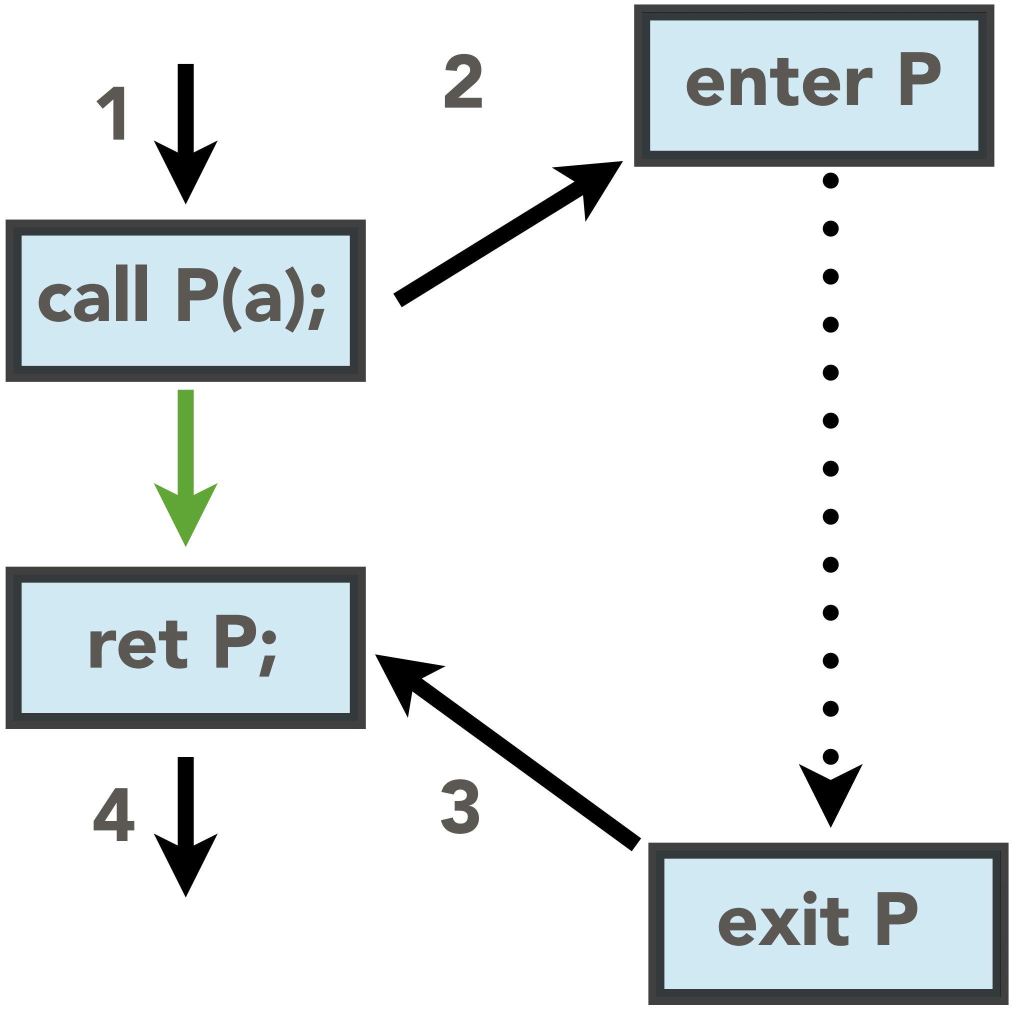
ret P;

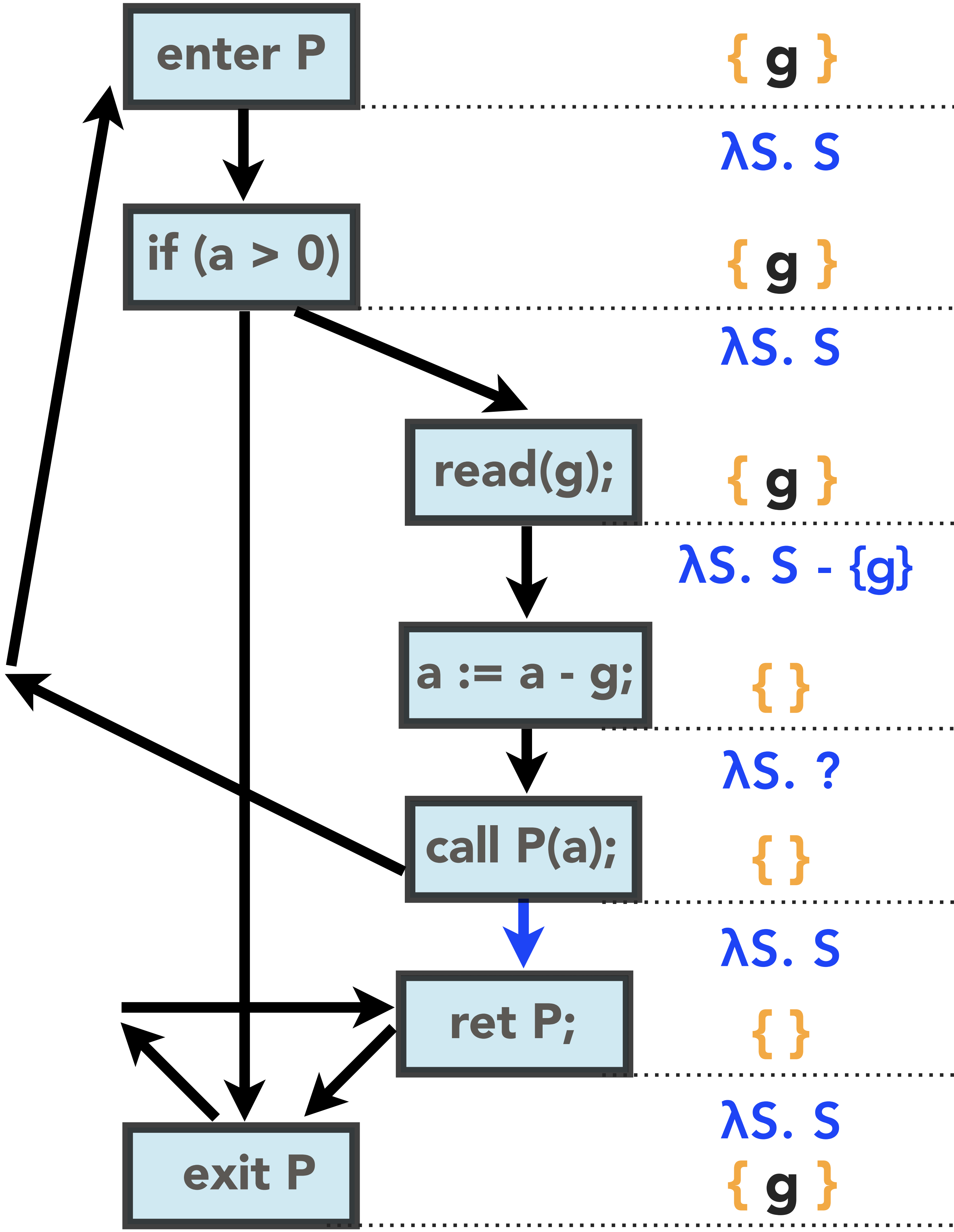
exit P

IFDS, POPL 1995

Meet over all **valid** paths

- Calls & Returns must match
- Enforced by **call** & **ret** nodes
- Track local variables with a **call-to-return edge**





enter P

$\{ g \}$

$\lambda S.\ S$

if (a > 0)

$\{ g \}$

$\lambda S.\ S$

read(g);

$\{ g \}$

$\lambda S.\ S - \{g\}$

a := a - g;

$\{ \}$

$\lambda S.\ ?$

call P(a);

$\{ \}$

$\lambda S.\ S$

ret P;

$\{ \}$

$\lambda S.\ S$

exit P

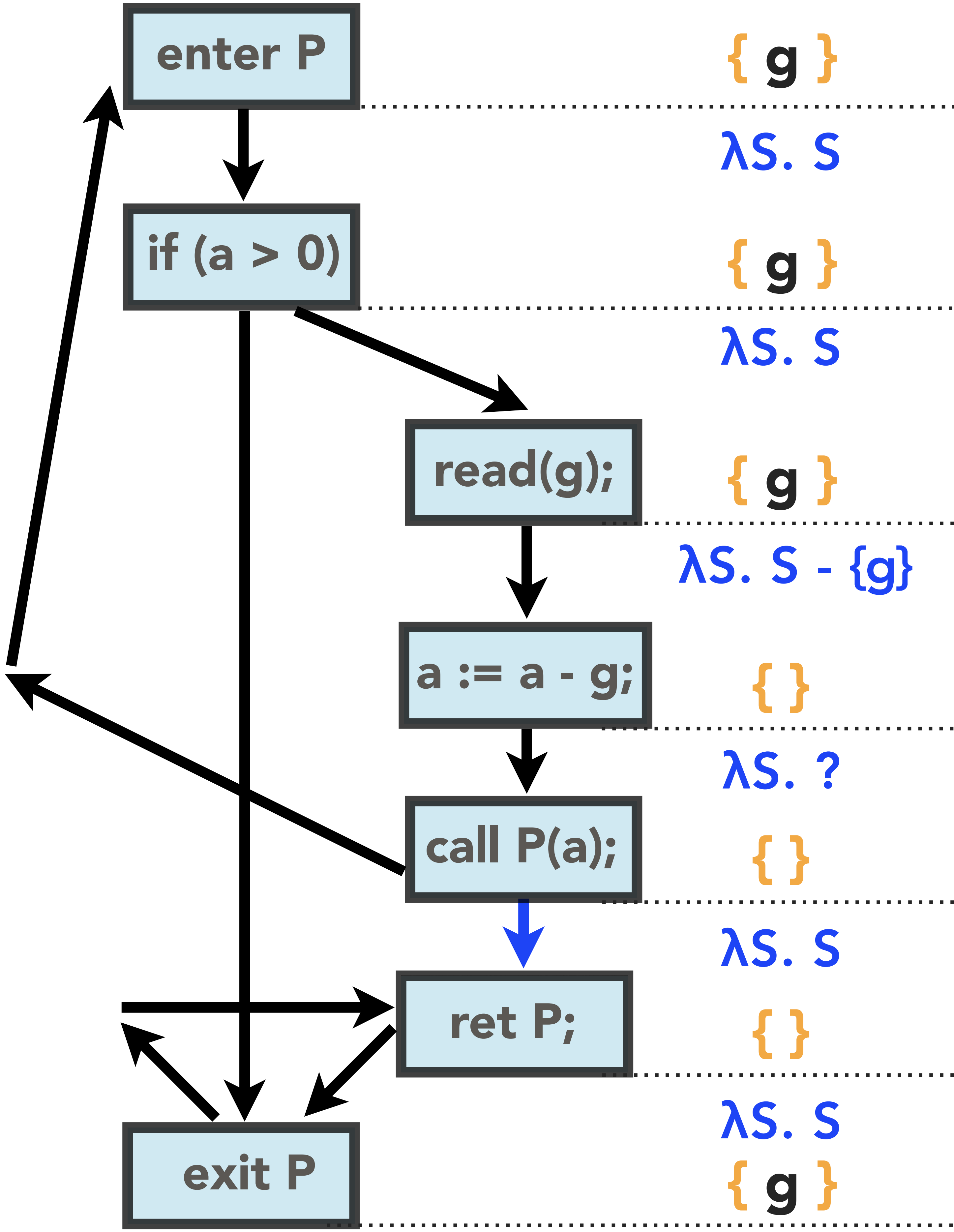
$\{ g \}$

The "supergraph"

```
int g;

void main(void) {
    int x;
    read(x);
    P(x);
}
```

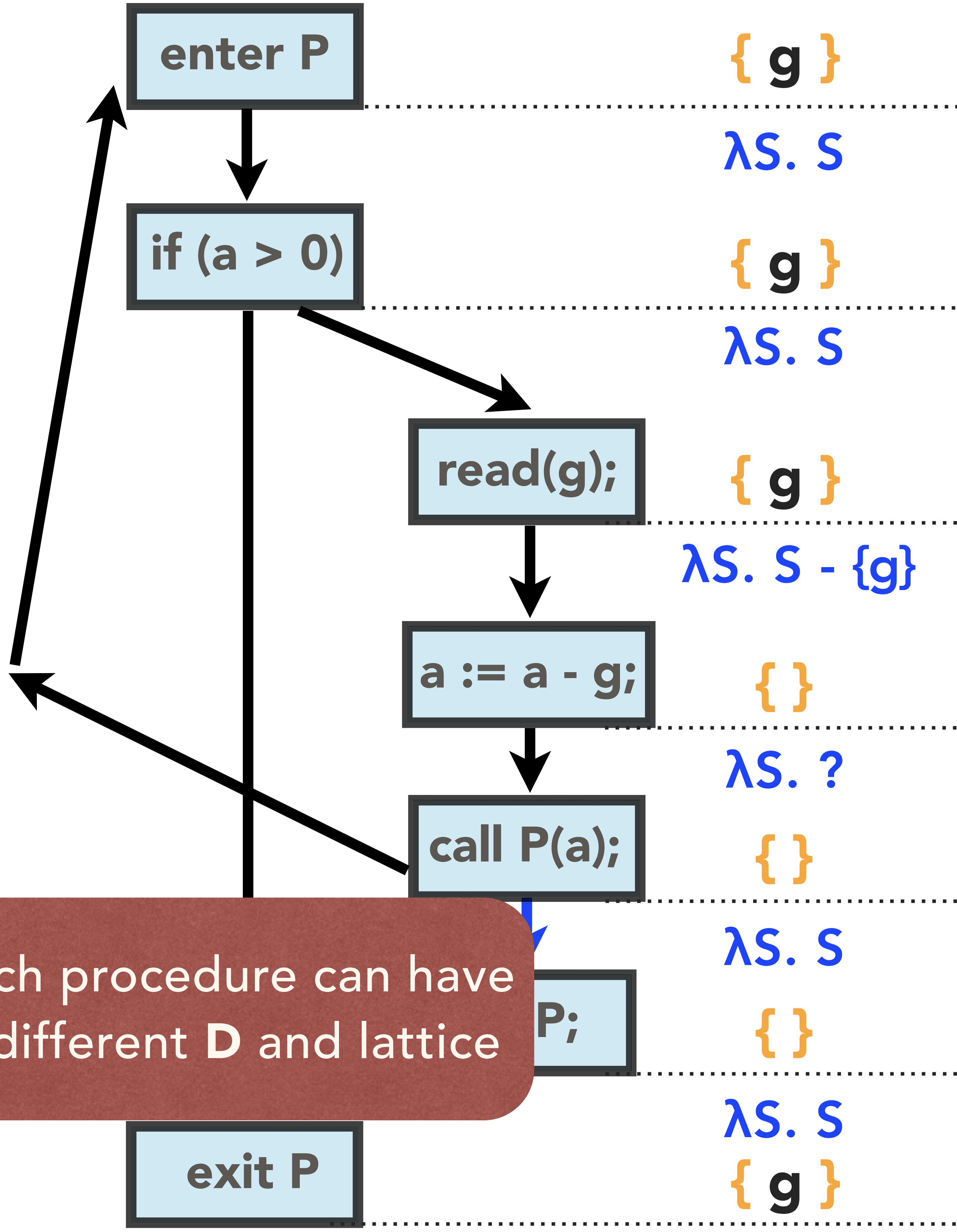
Adds similar CFGs for other procedures



The "supergraph"

```
int g;

void main(void) {
    int x;
    read(x);
    P(x);
}
```



enter P

if (a > 0)

read(g);

a := a - g;

call P(a);

P;

exit P

{ g }

λS. S

{ g }

λS. S

{ g }

λS. S - {g}

{ }

λS. ?

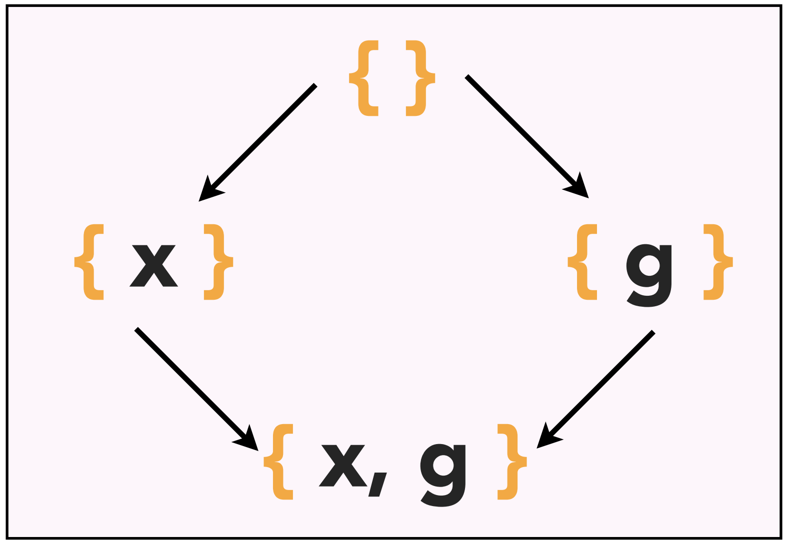
{ }

λS. S

{ }

λS. S

{ g }



Each procedure can have a different **D** and lattice

An IFDS problem instance

Interprocedural

Finite

Distributive

Subset

- G = a supergraph
- **D** = a finite set (determines a **lattice**)
- F = a set of **distributive** functions over the lattice
- M = a map from edges in G to functions in F
- $\sqcap$= meet operator on the lattice

A few "IFDS" problems

	D
Reaching definitions	All Variables
Available Expressions	All Expressions
Live Variable Analysis	All Variables
Possibly-Uninitialized Variables	All Variables
Type Analysis	Variables × Types

In general, apply "path function" to a subset of D

$$\sqcap \; fn\,(\,\dots\,(\,f1\,(\overset{D}{\boxed{\{\,g\,\}}}))\,\dots)$$

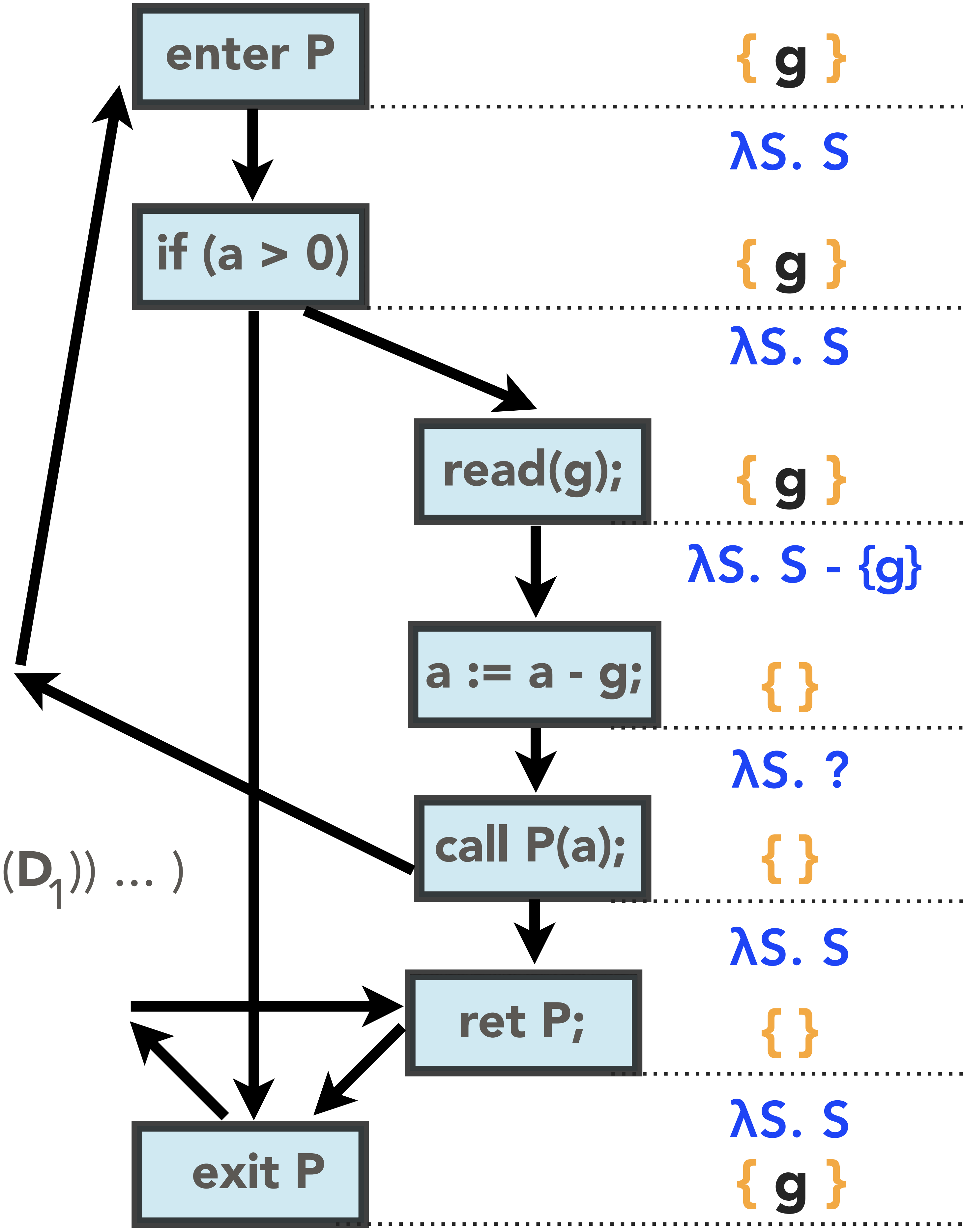
f1 … fn ∈ All Valid Paths

… since each **fi** is distributive

$$fi\,(D) = \{\, fi\,(\,D_1\,)\,,\,\dots\,,\,fi\,(\,D_k\,)\,\}$$

… therefore

$$fn\,(\,\dots\,(\,f1\,(D))\,\dots) = \{\, fn\,(\,\dots\,(\,f1\,(D_1))\,\dots\,)\,,\,\dots_k\,\}$$



$\bigsqcap$ fn (… (f1 (**D**)) …)

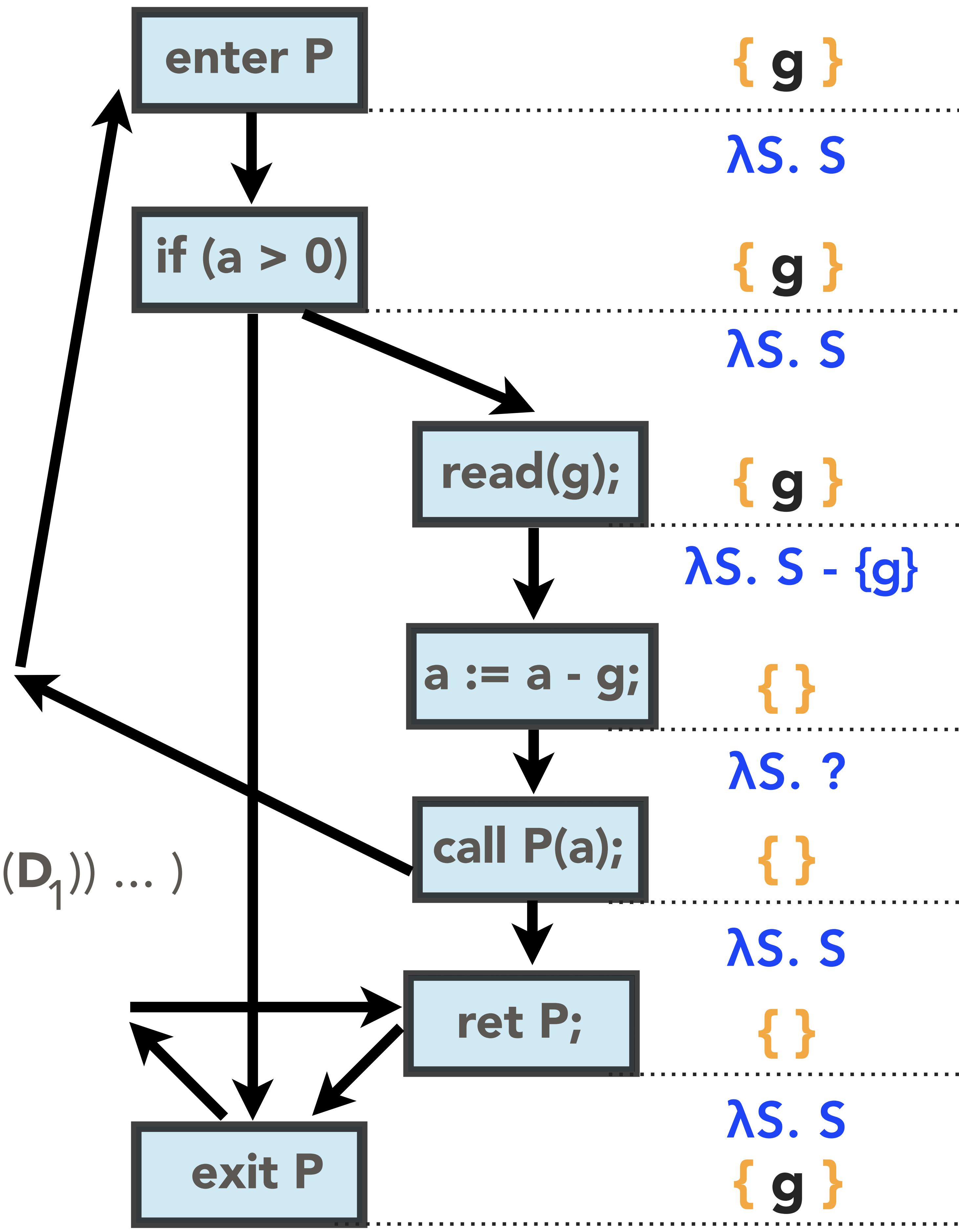
f1 … fn ∈ All Valid Paths

… since each fi is distributive

$fi(D) = \{ fi(D_1), \ldots, fi(D_k) \}$

… therefore

$fn(\ldots(f1(D))\ldots) = \{ fn(\ldots(f1(D_1))\ldots), \ldots_k \}$



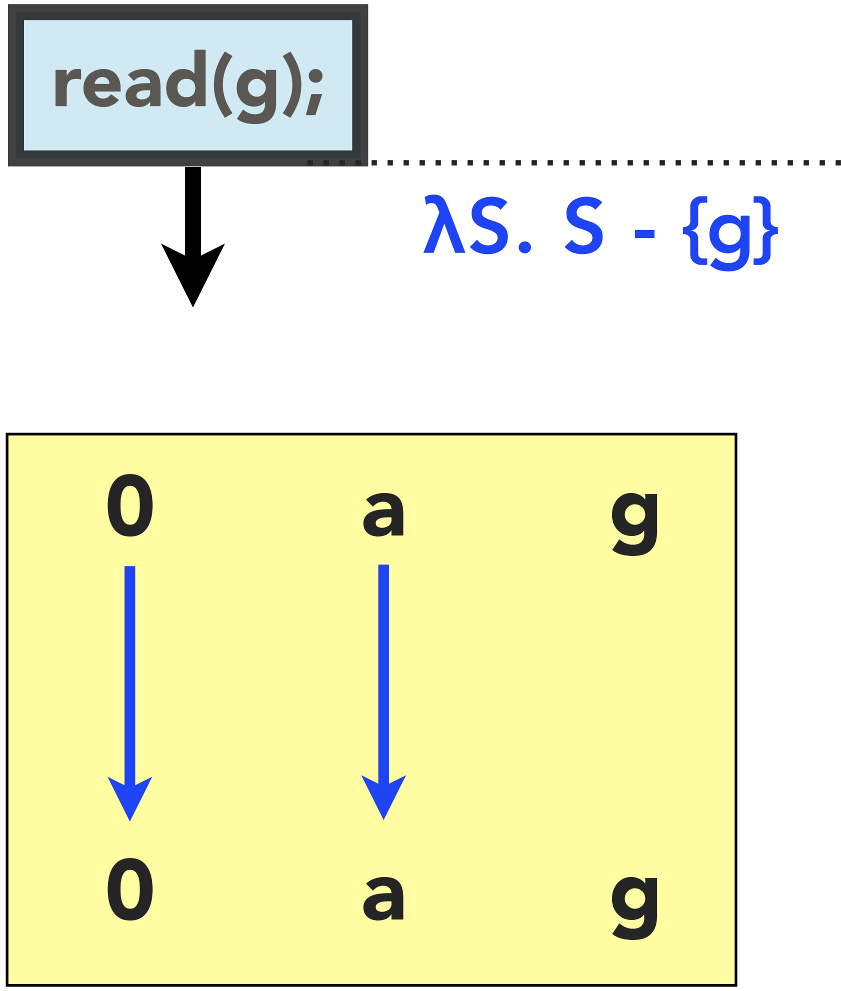
$$\bigsqcap_{f1 \ldots fn \in \text{All Valid Paths}} fn (\ldots (f1 (\mathbf{D})) \ldots)$$

… since each fi is distributive

$$fi (\mathbf{D}) = \{ fi (\mathbf{D}_1) , \ldots , fi (\mathbf{D}_k) \}$$

… therefore

$$fn (\ldots (f1 (\mathbf{D})) \ldots) = \{ fn (\ldots (f1 (\mathbf{D}_1)) \ldots) , \ldots_k \}$$



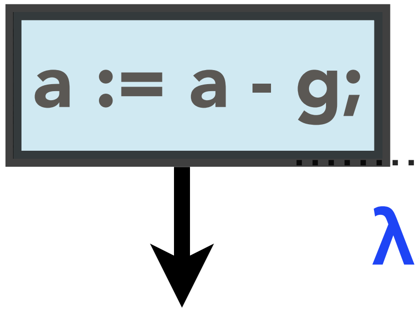
$$\bigsqcap_{f1 \;\ldots\; fn \;\in\; \text{All Valid Paths}} fn\,(\,\ldots\,(\,f1\,(\;\mathbf{D}\;))\,\ldots)$$

… since each **fi** is distributive

$$fi\,(\mathbf{D}) = \{\; fi\,(\,\mathbf{D}_1\,)\,,\,\ldots\,,\,fi\,(\,\mathbf{D}_k\,)\;\}$$

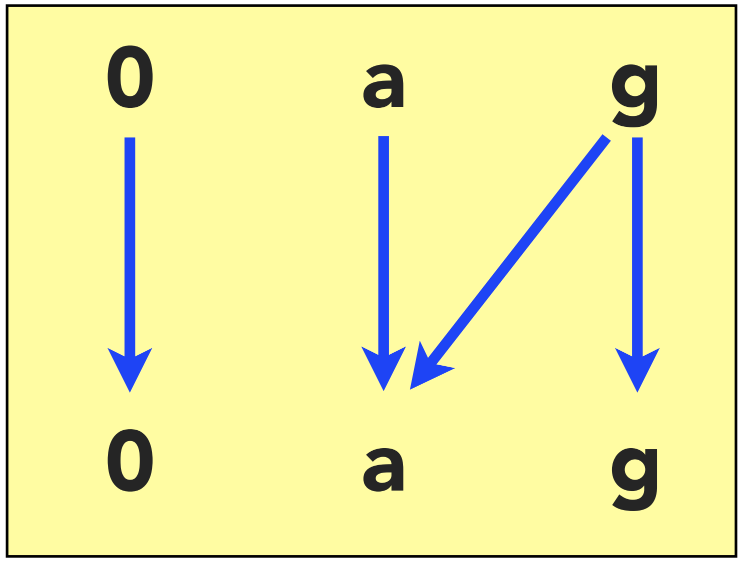
… therefore

$$fn\,(\,\ldots\,(\,f1\,(\mathbf{D}))\,\ldots) = \{\; fn\,(\,\ldots\,(\,f1\,(\mathbf{D}_1))\,\ldots\,)\,,\,\ldots_k\;\}$$



a := a - g;

λS.
if a ∈ S
or g ∈ S
then S ∪ {a}
else S - {a}



$$\underset{f1\ \ldots\ fn\ \in\ \text{All Valid Paths}}{\bigsqcap} fn\ (\ \ldots\ (\ f1\ (\ \mathbf{D}\ ))\ \ldots)$$

... since each fi is distributive

$$fi\ (\mathbf{D}) = \{\ fi\ (\ \mathbf{D}_1\ )\ ,\ \ldots\ ,\ fi\ (\ \mathbf{D}_k\ )\ \}$$

... therefore

$$fn\ (\ \ldots\ (\ f1\ (\mathbf{D}))\ \ldots) = \{\ fn\ (\ \ldots\ (\ f1\ (\mathbf{D}_1))\ \ldots\ )\ ,\ \ldots_k\ \}$$

For any f,

$$\mathbf{0} \longrightarrow \mathbf{0}$$

$$\mathbf{0} \longrightarrow \mathbf{y} \quad \text{if } f(0) = \mathbf{y}$$

$$\mathbf{x} \longrightarrow \mathbf{y} \quad \text{if } f(\mathbf{x}) = \mathbf{y}$$

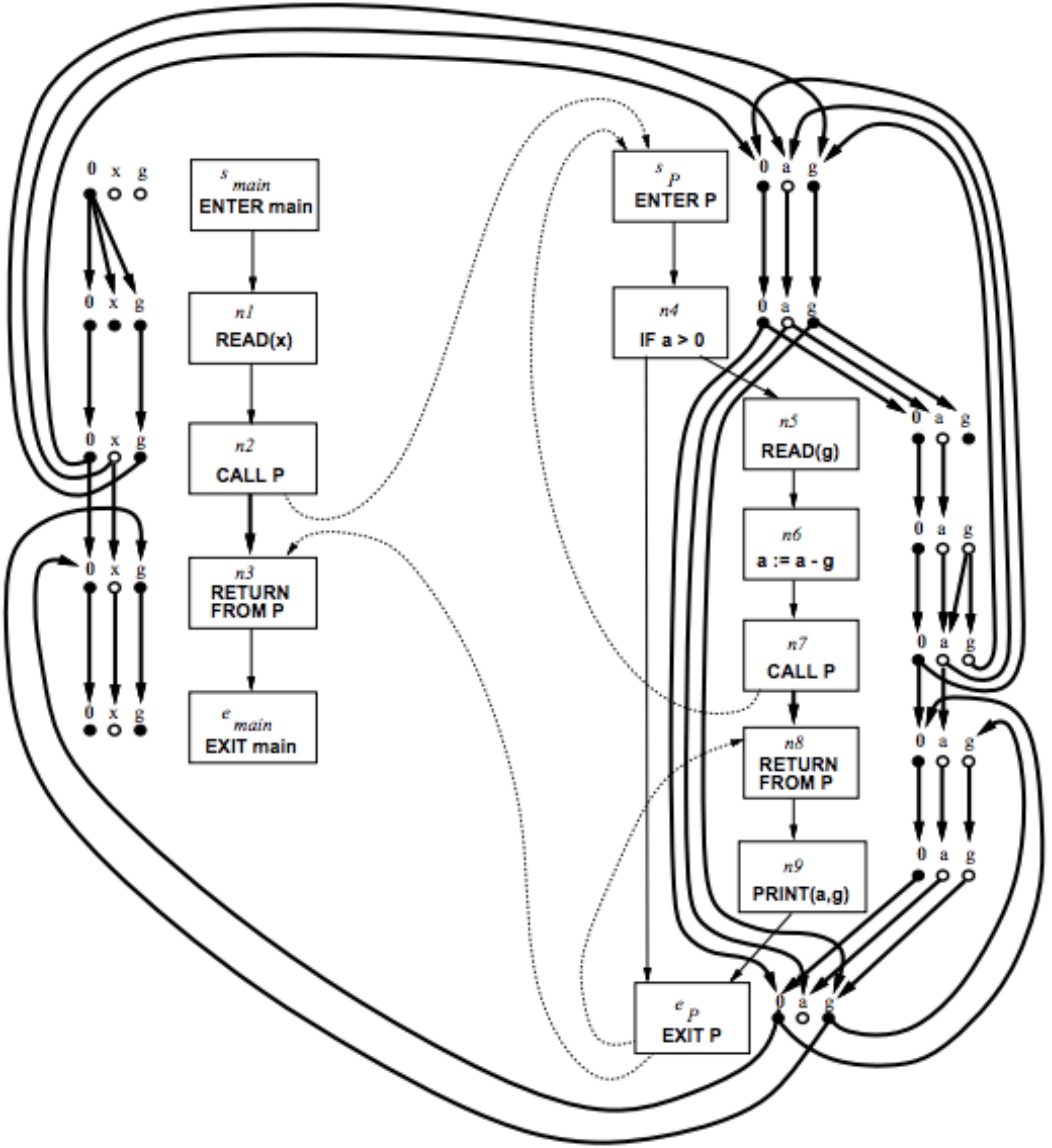
$$\text{and } f(0) \neq \mathbf{y}$$

Exploded supergraph

```
int g;

void main(void) {
    int x;
    read(x);
    P(x);
}

void P(int a) {
    if (a > 0) {
        read(g);
        a := a - g;
        P(a);
    }
}
```



"Tabulation" Algorithm

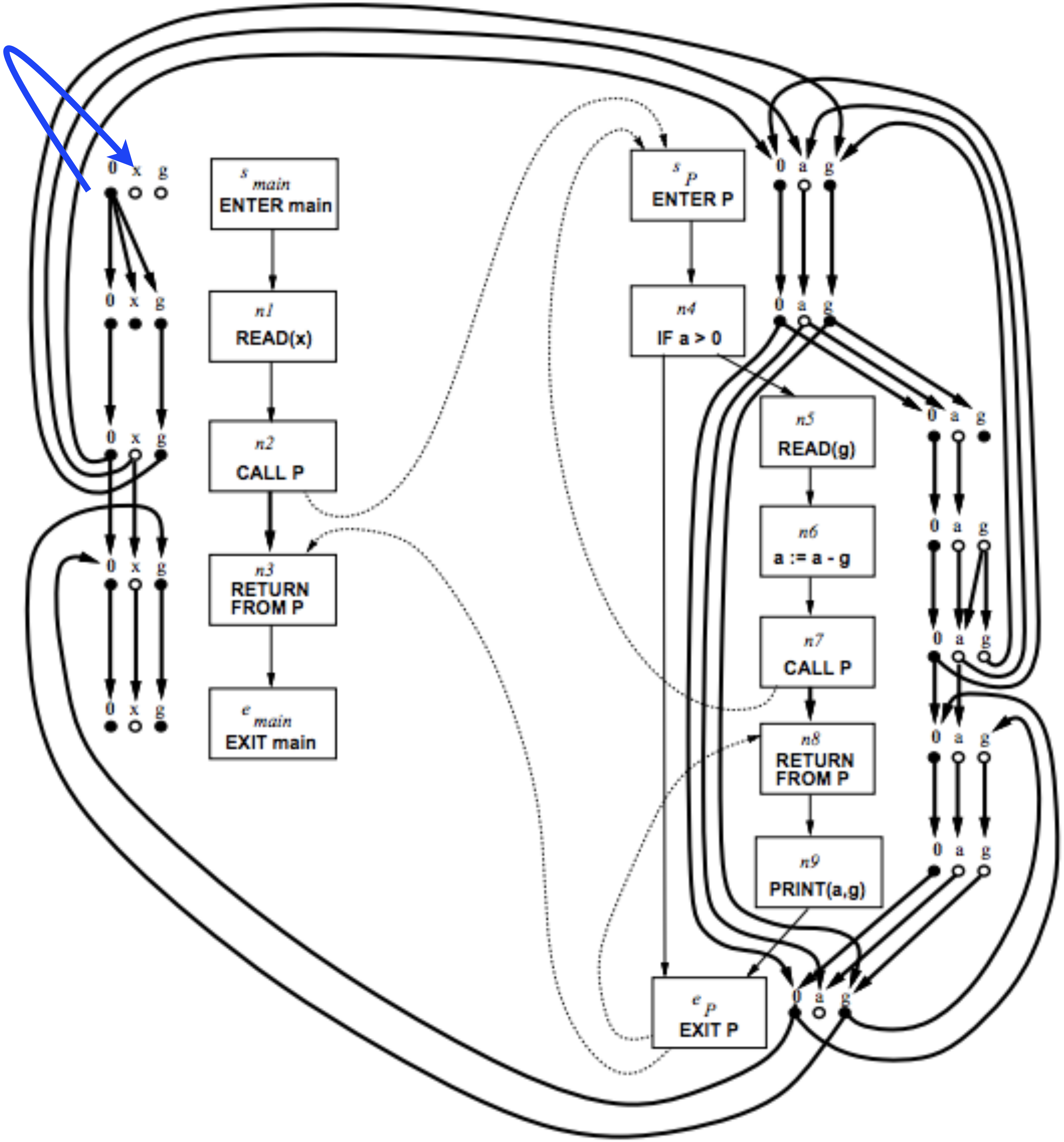
1. keep a worklist of **Path Edges**
    - (suffixes of valid paths)
2. build set of **Summary Edges**
    - (side effects of a procedure call)
3. result = meet over valid paths

Init

(lines 1-4)

Path Edge

main 0 → main 0



Summary Edge:

Case $n \notin$ Call, $n \notin$ Exit

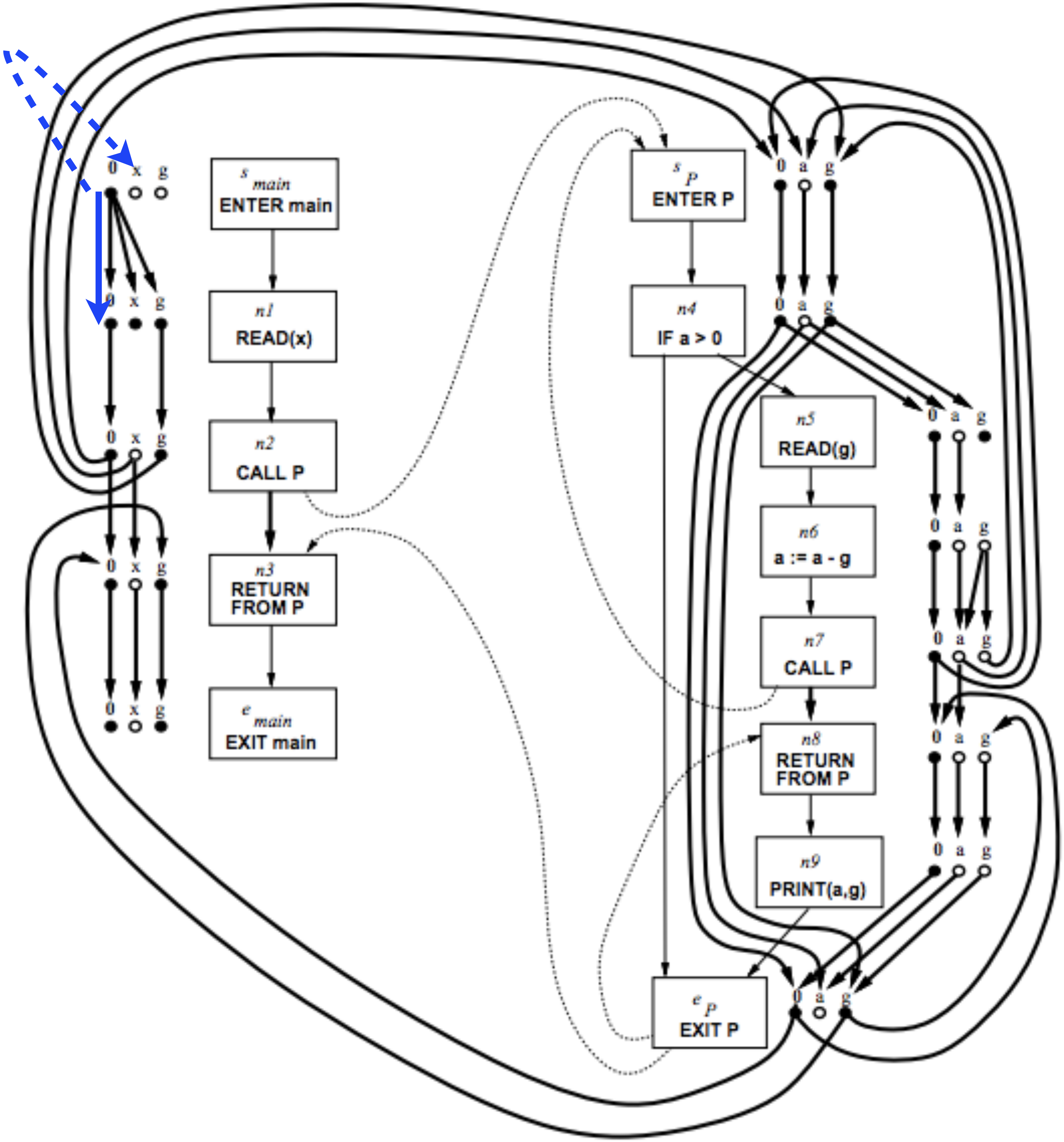
(lines 31-33)

Path Edge

main 0 → main 0

main 0 → n1 {x,g}

Summary Edge:



Case n ∉ Call, n ∉ Exit

(lines 31-33)

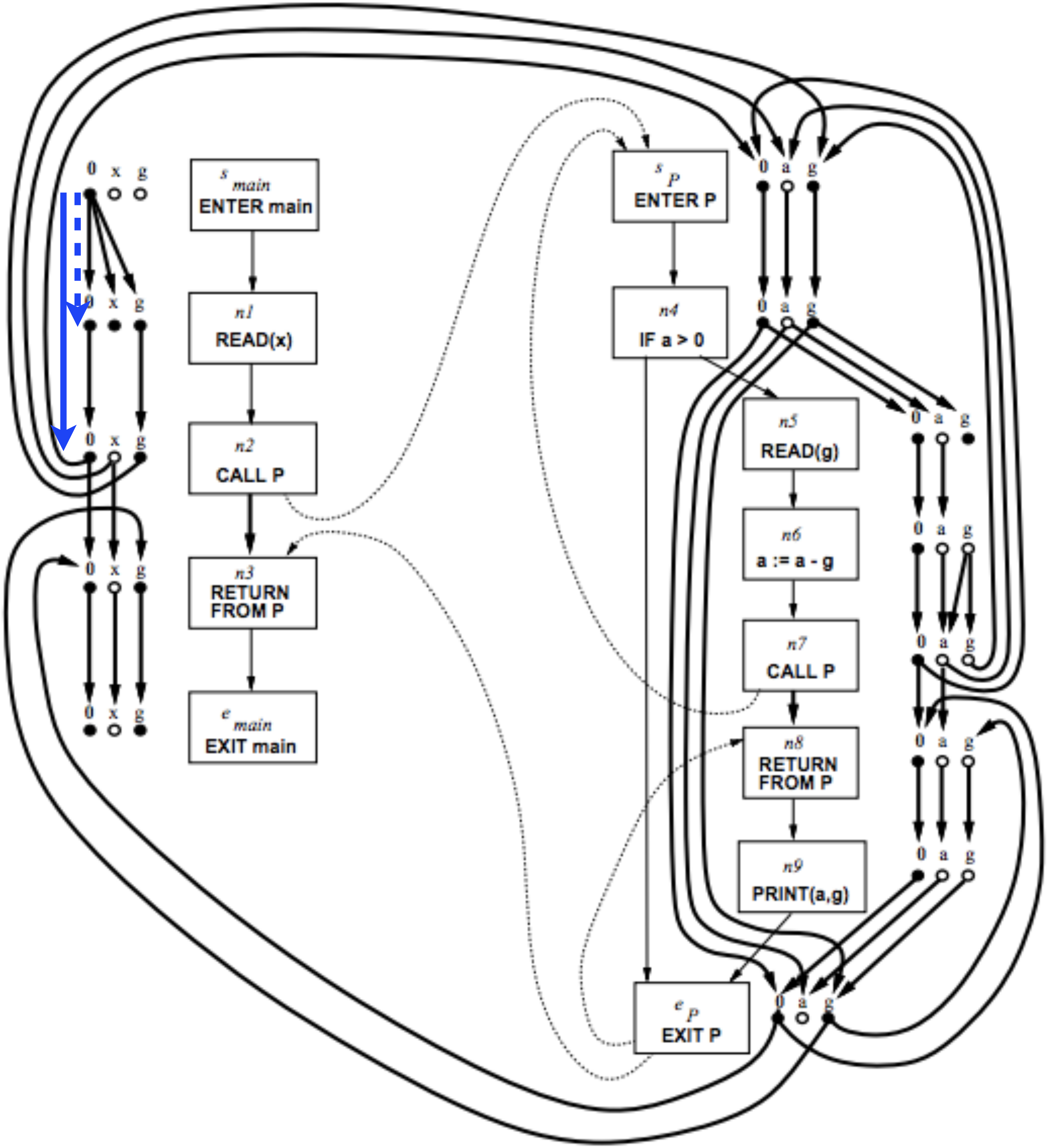
Path Edge

main 0 → main 0

main 0 → n1 {x,g}

main 0 → n2 {g}

Summary Edge:



Case n ∈ Call

(lines 13-20)

Path Edge

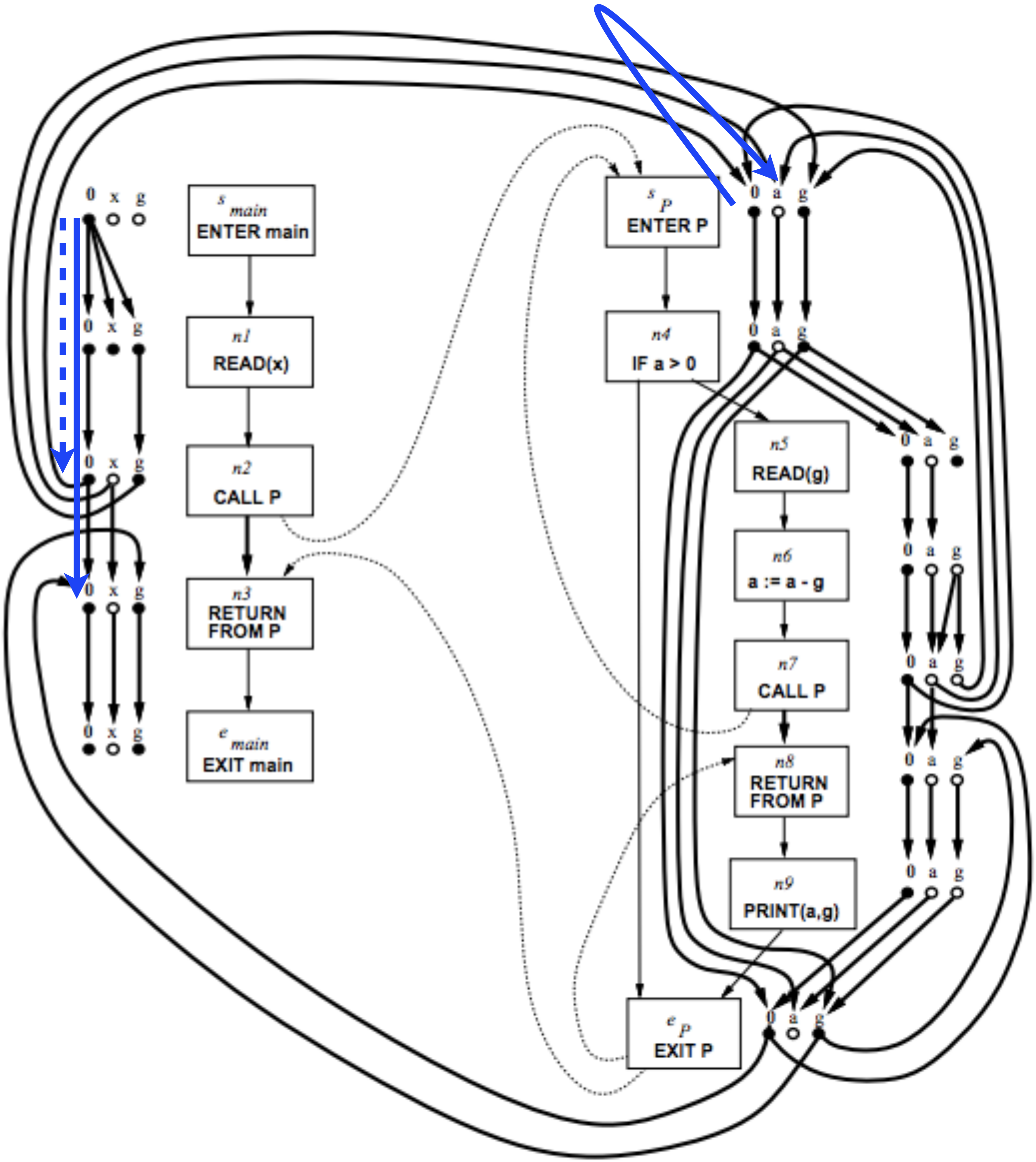
main 0 → main 0

main 0 → n1 {x,g}

main 0 → n2 {g}

sp {g} → sp {g}

main 0 → n3 {g}



Summary Edge:

Case n ∉ Call, n ∉ Exit

(lines 31-33)

Path Edge

main 0 → main 0

main 0 → n1 {x,g}

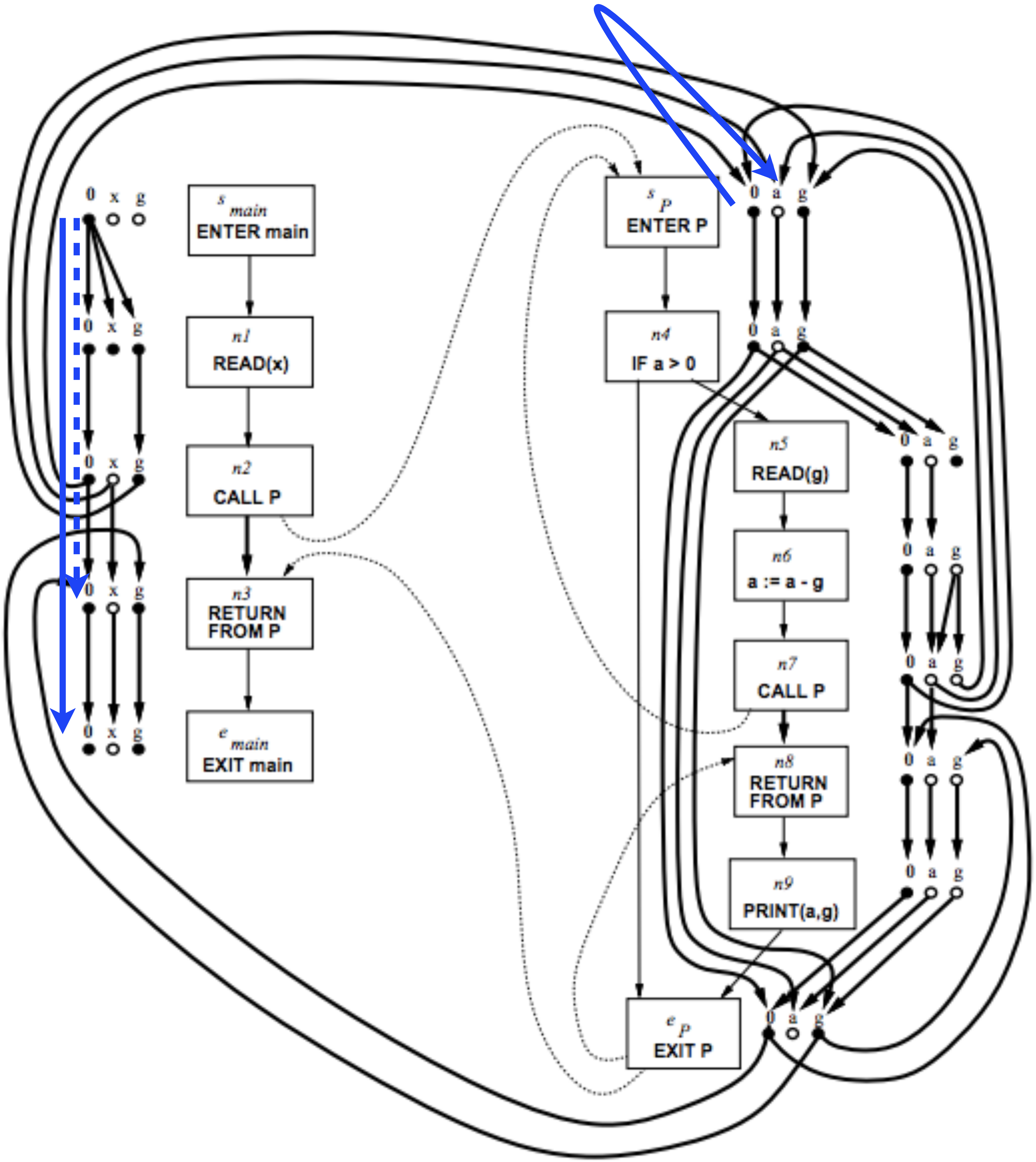
main 0 → n2 {g}

sp {g} → sp {g}

main 0 → n3 {g}

main 0 → exit {g}

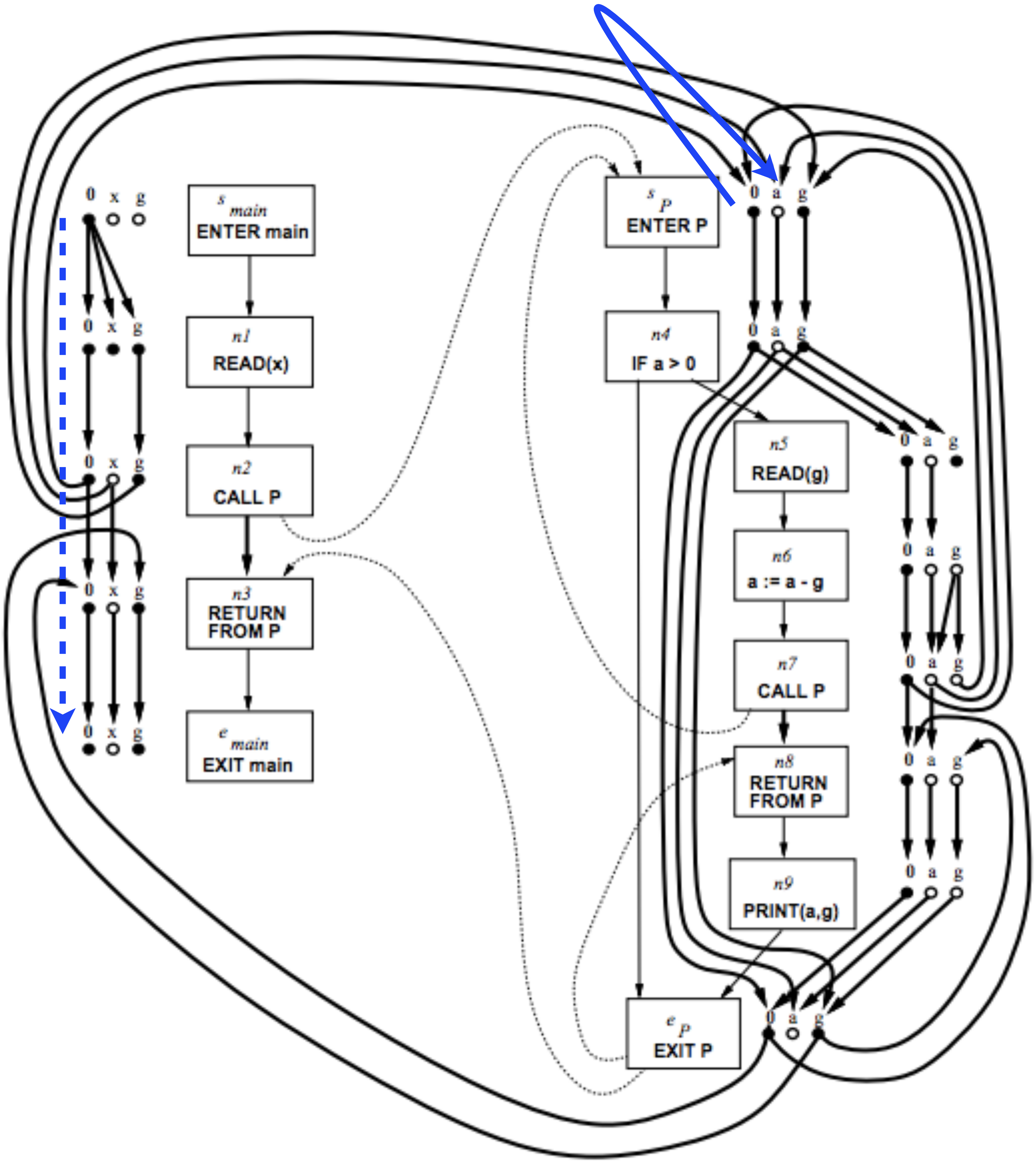
Summary Edge:



Case $n \notin$ Call, $n \notin$ Exit
(lines 31-33)

Path Edge

main 0 → main 0
main 0 → n1 {x,g}
main 0 → n2 {g}
sp {g} → sp {g}
main 0 → n3 {g}
main 0 → exit {g}

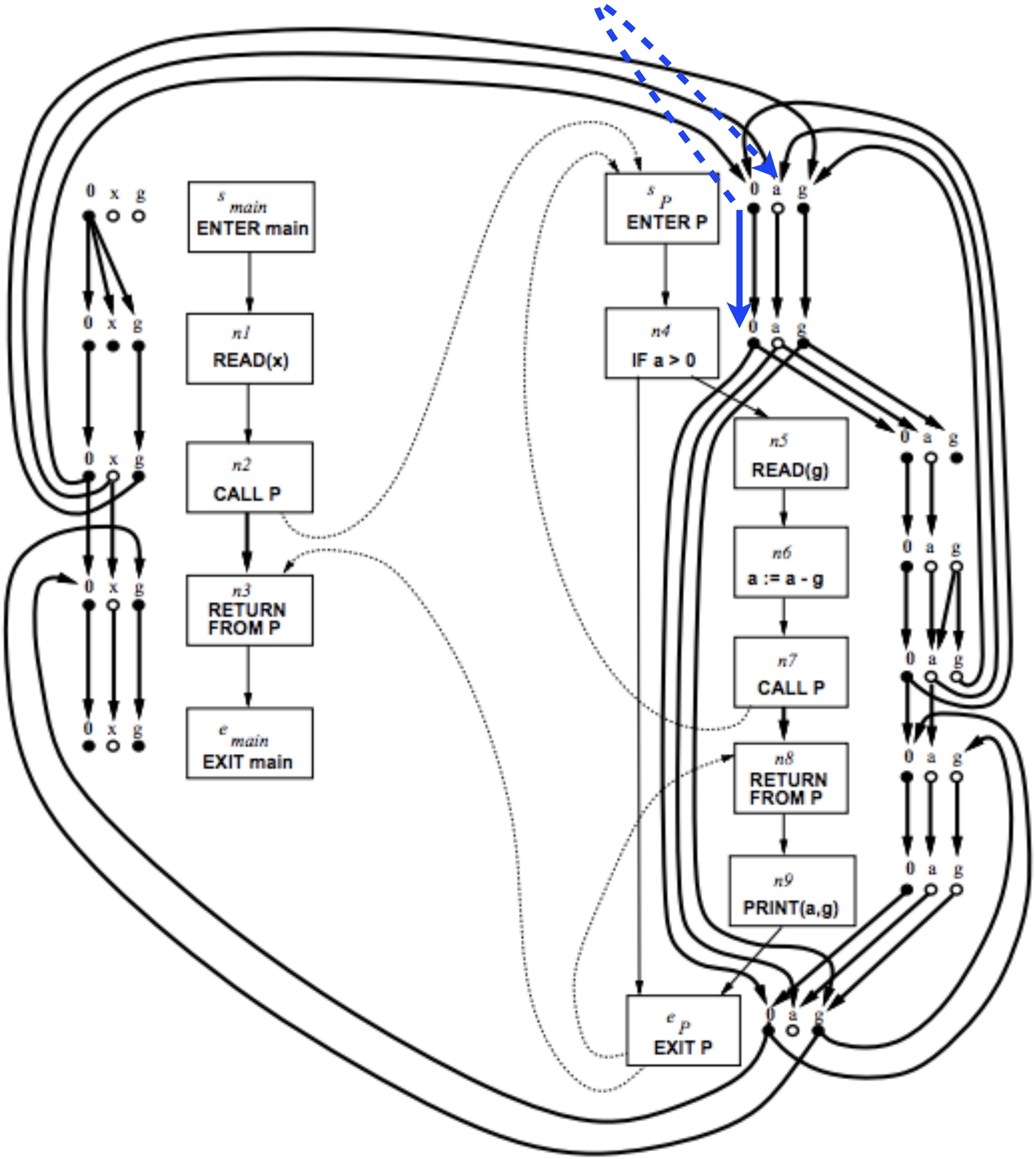


Summary Edge:

Case n ∉ Call, n ∉ Exit
(lines 31-33)

Path Edge

main 0 → main 0
main 0 → n1 {x,g}
main 0 → n2 {g}
sp {g} → sp {g}
main 0 → n3 {g}
main 0 → exit {g}
sp {g} → n4 {g}



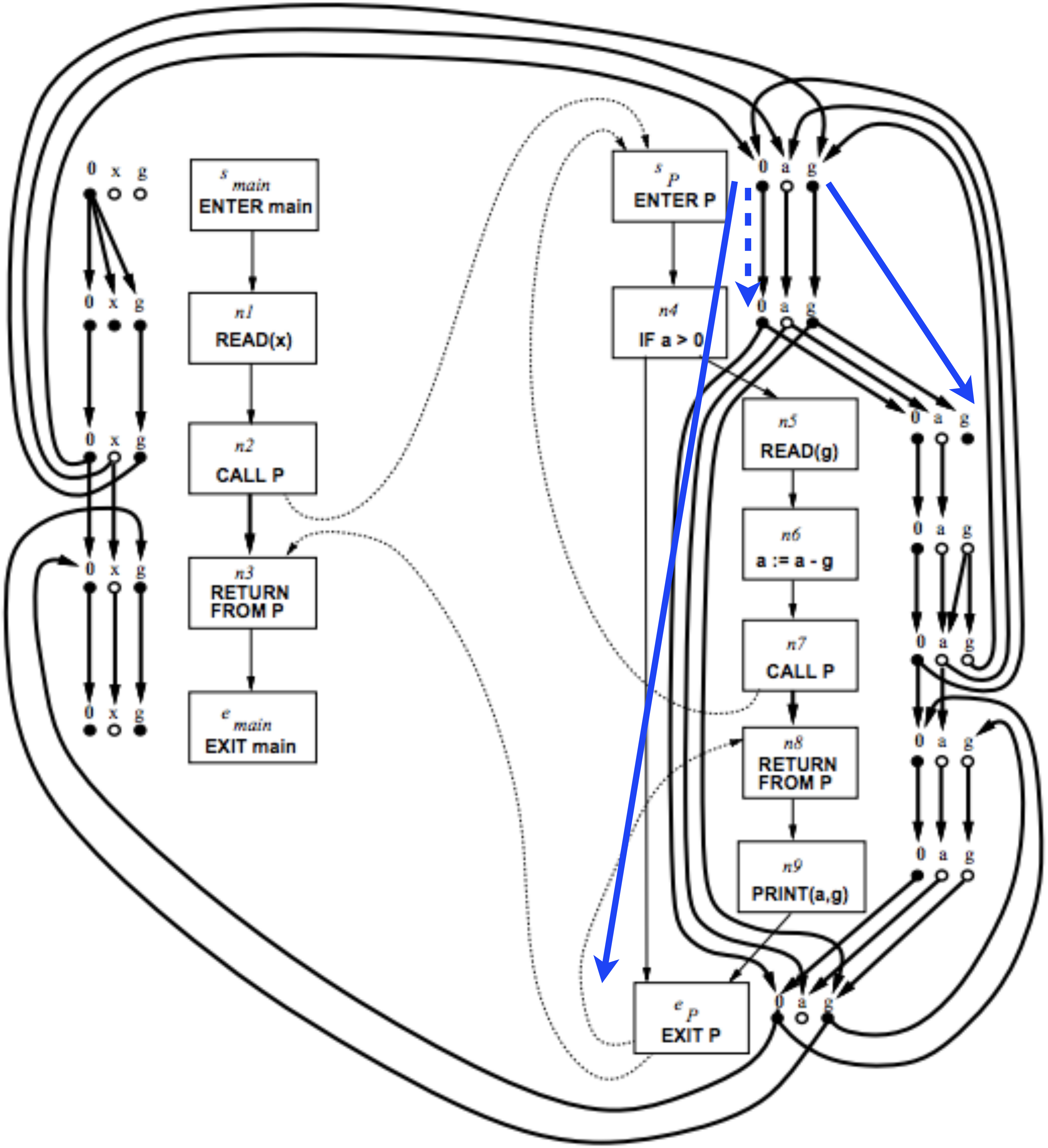
Summary Edge:

Case n ∉ Call, n ∉ Exit
(lines 31-33)

Path Edge

main 0 → main 0
main 0 → n1 {x,g}
main 0 → n2 {g}
sp {g} → sp {g}
main 0 → n3 {g}
main 0 → exit {g}
sp {g} → n4 {g}
sp {g} → exit {g}
sp {g} → n5 {g}

Summary Edge:



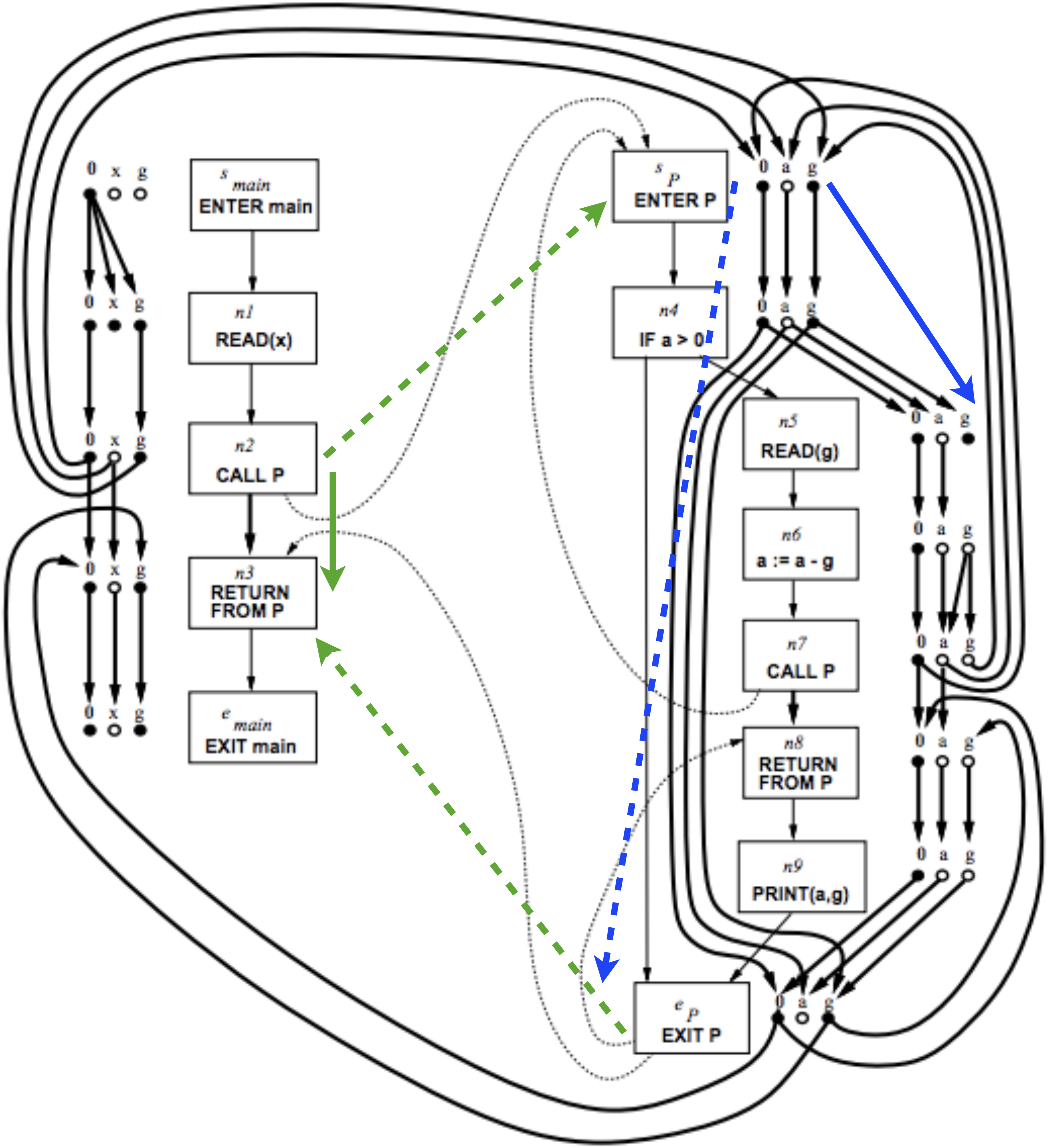
Case n ∈ Exit
(lines 21-25)

Path Edge

main 0 → main 0
main 0 → n1 {x,g}
main 0 → n2 {g}
sp {g} → sp {g}
main 0 → n3 {g}
main 0 → exit {g}
sp {g} → n4 {g}
sp {g} → exit {g}
sp {g} → n5 {g}

Summary Edge:

n2 {g} → n3 {g}



Case n ∈ Exit
(lines 25-32)

Path Edge

main 0 → main 0

main 0 → n1 {x,g}

main 0 → n2 {g} ★

sp {g} → sp {g}

main 0 → n3 {g}

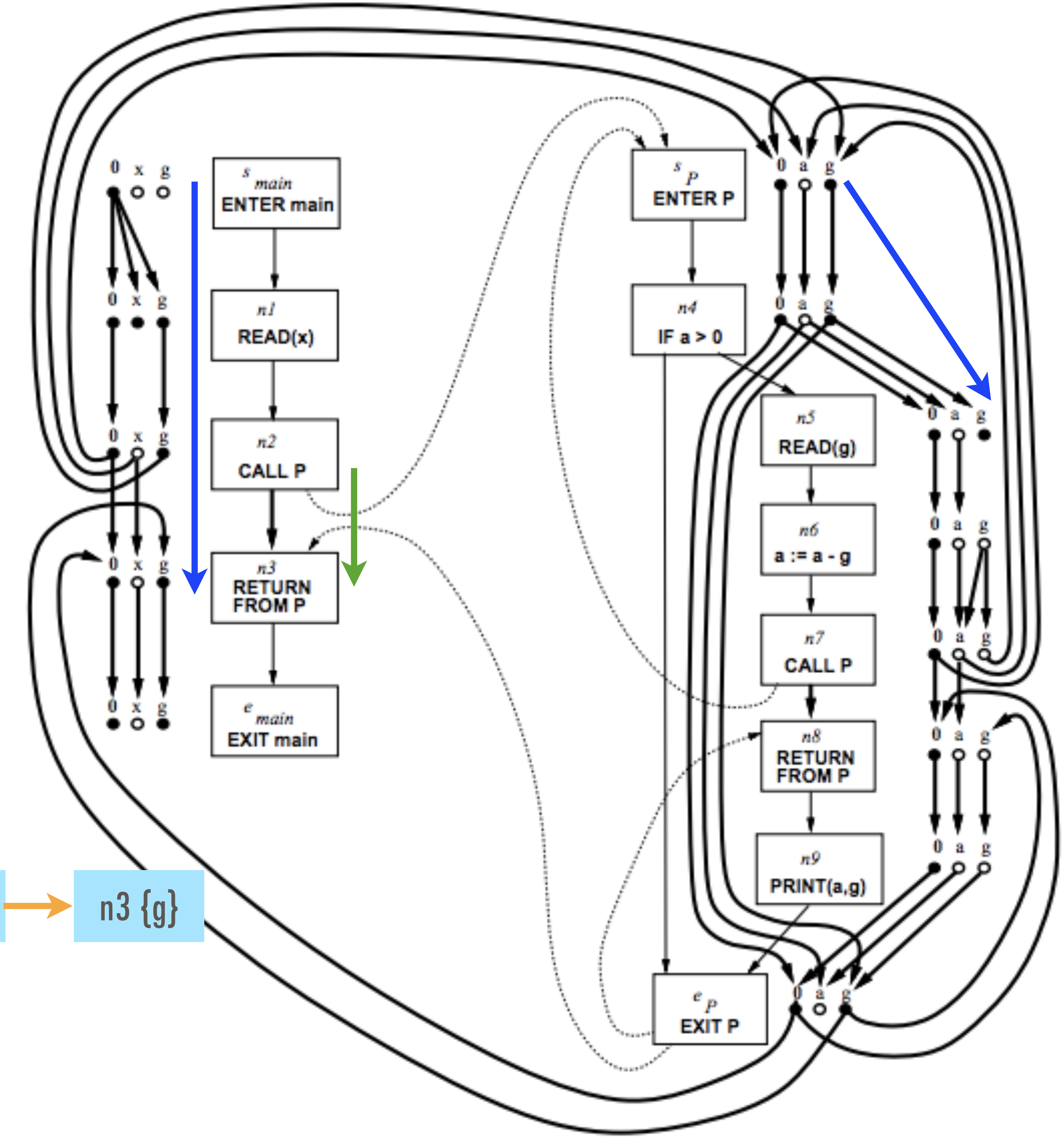
main 0 → exit {g}

sp {g} → n4 {g}

sp {g} → exit {g}

sp {g} → n5 {g}

main 0 → n3 {g}



Summary Edge:

n2 {g} → n3 {g}

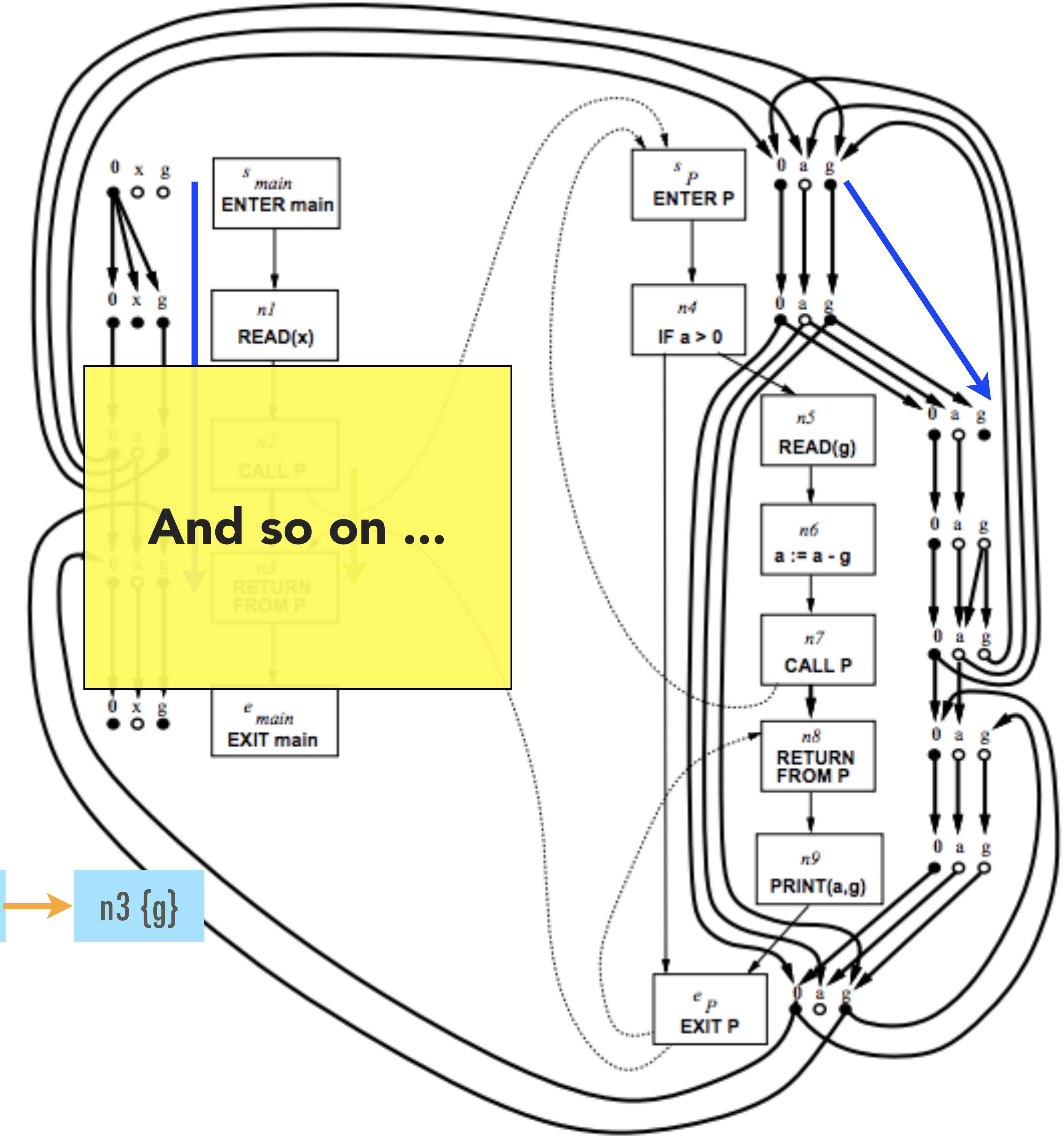
Case n ∈ Exit
(lines 25-32)

Path Edge

main 0 → main 0
main 0 → n1 {x,g}
main 0 → n2 {g}
sp {g} → sp {g}
main 0 → n3 {g}
main 0 → exit {g}
sp {g} → n4 {g}
sp {g} → exit {g}
sp {g} → n5 {g}
main 0 → n3 {g}

Summary Edge:

n2 {g} → n3 {g}



And so on …

Algorithm II

- "4" ways to find **Path Edges**

  1. **call** edge

  2. **return** edge / **Summary Edge**

  3. normal edge

Running Time

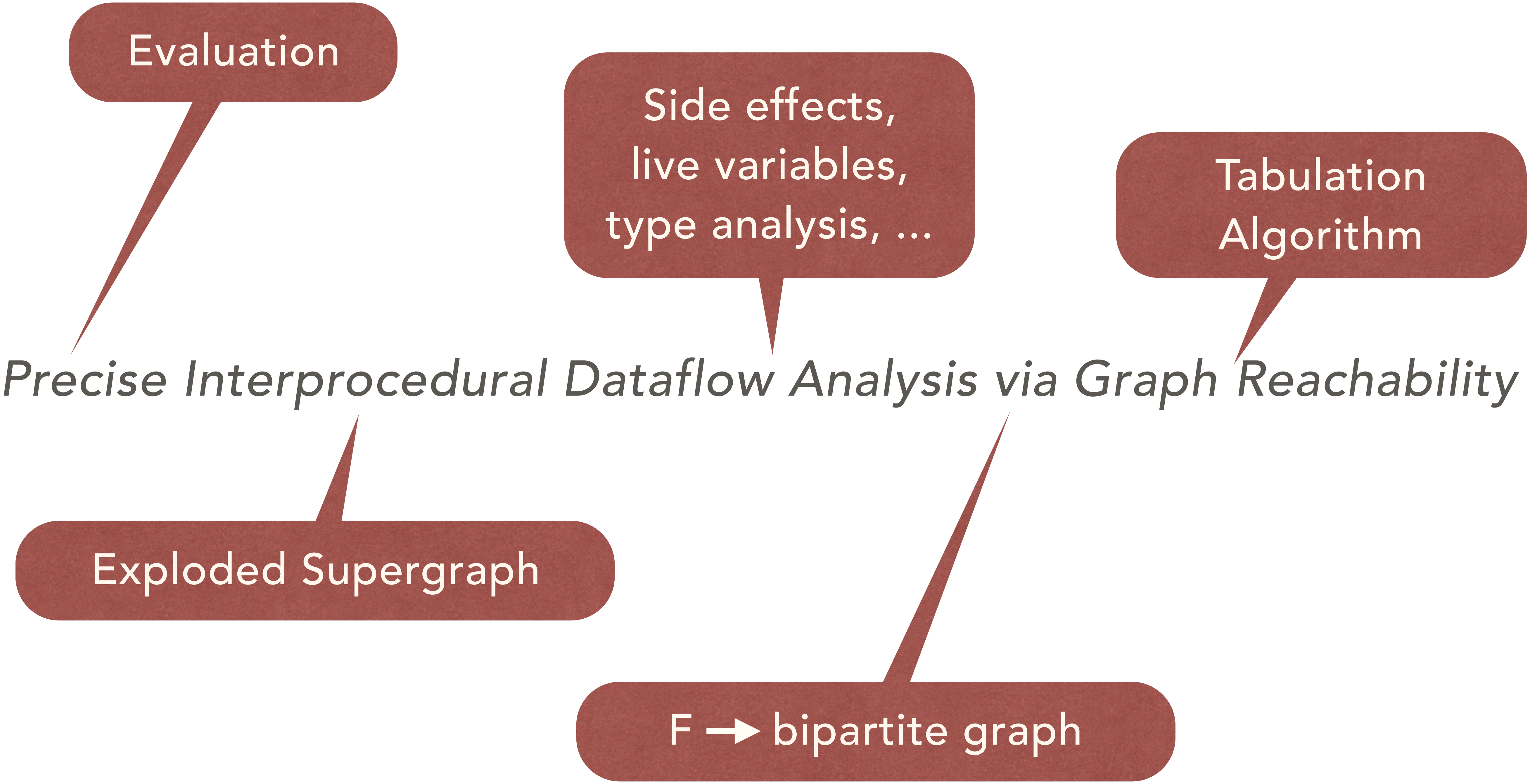
- **E** supergraph edges to explore
- **D** sources to explore from
- $\mathbf{D^2}$ exploded edges for each edge

Class of F	Running Time
Distributive	$O(ED^3)$
h-sparse	$O(\text{Call } D^3 + hED^2)$
Locally Separable	$O(ED)$

Evaluation

	supergraph stats					exploded supergraph stats		
Program	**# lines**	**# proc.**	**# calls**	**# nodes**	**# edges**	**D**	**# n++**	**# e++**
struct-beauty	897	36	214	2188	2860	90	184k	221k
C-parser	1224	48	78	1637	1992	70	104k	112k
ratfor	1345	52	266	2239	2991	87	180k	218k
twig	2388	81	221	3692	4439	142	492k	561k

Program	naive time (s)	naive # null	ifds time (s)	ifds # null
struct-beauty	1.58	583	4.83 (+ 3.25)	543 (- 40)
C-parser	0.54	127	0.7 (+ 0.16)	11 (- 116)
ratfor	1.46	998	3.15 (+ 1.69)	894 (- 104)
twig	5.04	775	5.45 (+ 0.41)	767 (- 8)



Evaluation

Side effects,
live variables,
type analysis, …

Tabulation
Algorithm

Precise Interprocedural Dataflow Analysis via Graph Reachability

Exploded Supergraph

F → bipartite graph

Discussion

- What static analysis problems are / are not IFDS ?
- The uninitialized variables problem is **cubic**

  in the # global variables, even if these are rarely used.

  Can we avoid this overhead?
- Could we allow a (restricted) GOTO?
- Can we add more context-sensitivity?

  (Naeem, Lhoták, Rodriguez; CC'10)

Influence

- WALA
- SOOT (Bodden; SOAP '12)
- FLIX (Madsen, Yee, Lhoták; PLDI 2016)
- FlowDroid (Arzt, Rasthofer, Fritz, Bodden, Bartel,
  Klein, Le Traon, Octeau, McDaniel; PLDI 2014)

END