Lecture 9

Parametric Static Analysis (2)

Hakjoo Oh

2016 Fall

Parametric Static Analysis

- $\boldsymbol{P} \in \mathbb{P}$: a program to analyze
- $\mathbb{Q}_{\boldsymbol{P}}$: a set of queries in $\boldsymbol{P}$
- $\mathbb{J}_{\boldsymbol{P}}$: a set of program components
- The parameter space $(\boldsymbol{\mathcal{A}_P}, \sqsubseteq)$:

$$\boldsymbol{a} \in \boldsymbol{\mathcal{A}_P} = \{\mathbf{0}, \mathbf{1}\}^{\mathbb{J}_{\boldsymbol{P}}}$$

  with $\boldsymbol{a} \sqsubseteq \boldsymbol{a'} \iff \forall \boldsymbol{j} \in \mathbb{J}_{\boldsymbol{P}}.\ \boldsymbol{a_j} \leq \boldsymbol{a'}_{\boldsymbol{j}}$.
- The parametric static analysis:

$$\boldsymbol{F_P} : \boldsymbol{\mathcal{A}_P} \to \wp(\mathbb{Q}_{\boldsymbol{P}}).$$

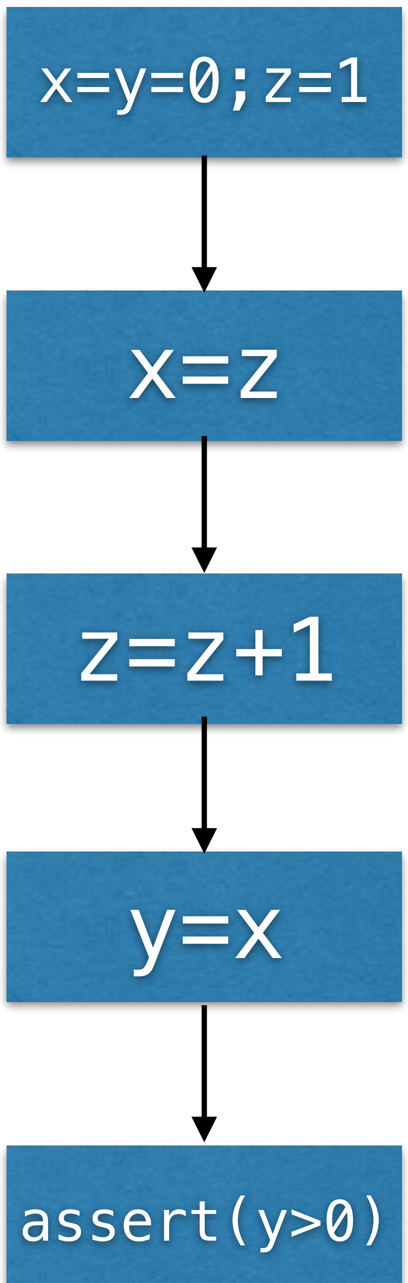
- Assume the monotonicity:

$$\boldsymbol{a} \sqsubseteq \boldsymbol{a'} \implies \boldsymbol{F_P}(\boldsymbol{a}) \subseteq \boldsymbol{F_P}(\boldsymbol{a}).$$

Parametric Analysis Problems

- Find $a \in \mathcal{A}_P$ such that
  - $F_P(a) = F_P(1)$ and
  - $\{a' \sqsubseteq a \mid \hat{F}_P(a) = \hat{F}_P(a')\} = \{a\}$.
  - "Learning minimal abstractions". POPL'11.
- Find an abstraction $a \in \mathcal{A}_P$ such that
  - the precision of $F_P(a)$ is close to that of $F_P(1)$, and
  - the cost of $F_P(a)$ is close to that of $F_P(0)$.
  - "Selective context-sensitivity guided by impact pre-analysis". PLDI'14.
  - "Learning a strategy to adapt a program analysis via bayesian optimization". OOPSLA'15.
  - "Abstractions from Tests". POPL'12
- Find the set R of all provable queries: i.e., $R = F_P(1)$.
  - "On abstraction refinement for program analyses in Datalog". PLDI'14.

Parametric Flow-Sensitivity



x	[0,0]
y	[0,0]
z	[1,1]

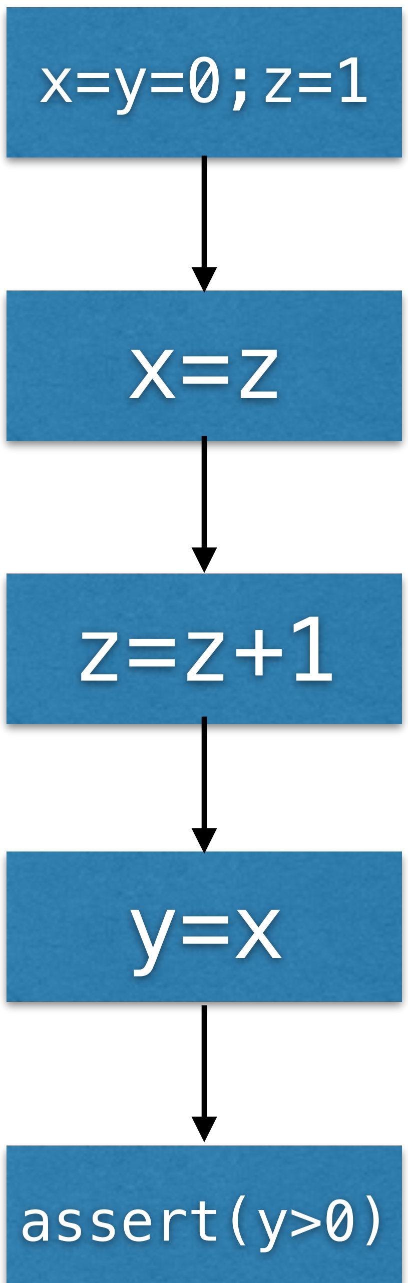
x	[1,1]
y	[0,0]
z	[1,1]

x	[1,1]
y	[0,0]
z	[2,2]

x	[1,1]
y	[1,1]
z	[2,2]

precise but costly

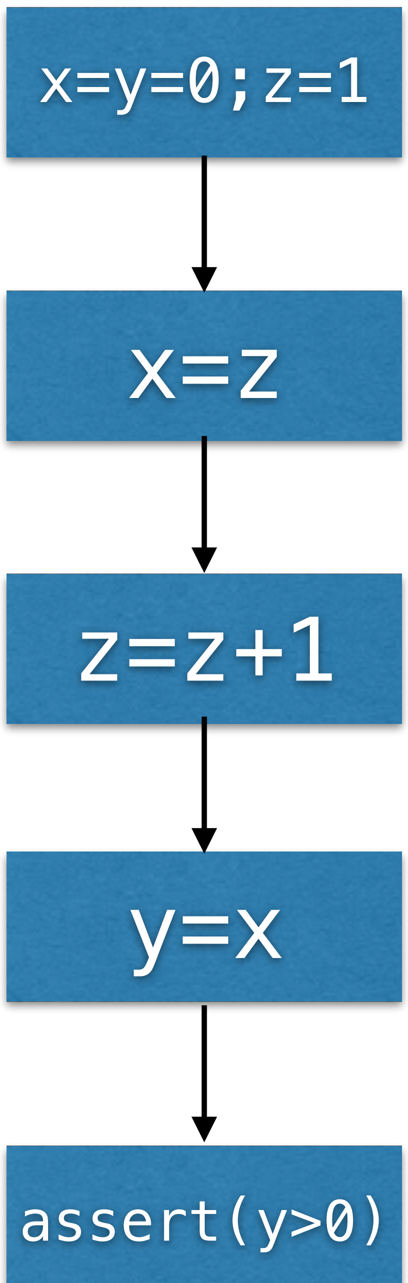
Parametric Flow-Sensitivity



x	$[0,+\infty]$
y	$[0,+\infty]$
z	$[1,+\infty]$

cheap but imprecise

Parametric Flow-Sensitivity



FS : {x,y}

x	[0,0]
y	[0,0]

x	[1,+∞]
y	[0,0]

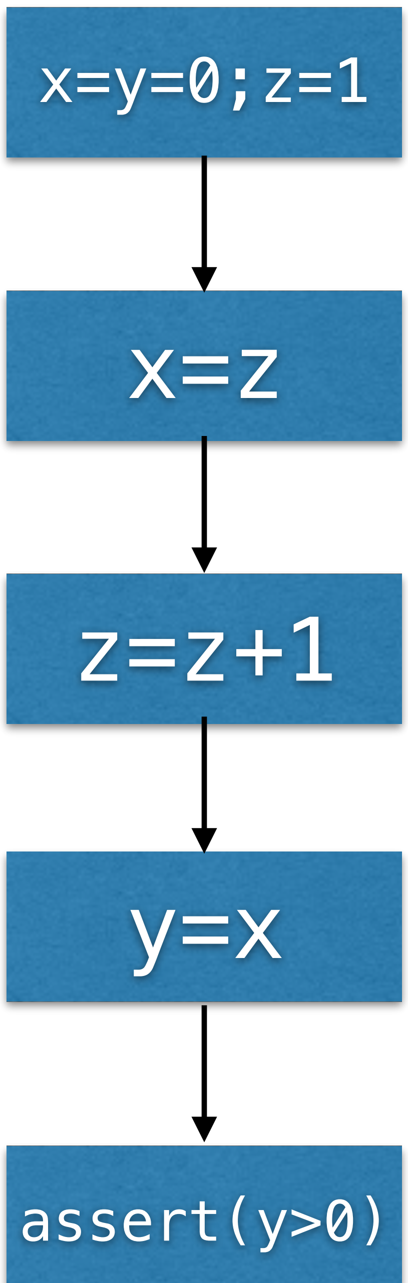
x	[1,+∞]
y	[0,0]

x	[1,+∞]
y	[1,+∞]

FI : {z}

z	[1,+∞]

Parametric Flow-Sensitivity



FS : {y,z}

y	[0,0]
z	[1,1]

y	[0,0]
z	[1,1]

y	[0,0]
z	[2,2]

y	[0,+∞]
z	[2,2]

fail to prove

FI : {x}

x	[0,+∞]

Finding a good abstraction is challenging

- Intractably large space, if not infinite
  - 2^{Var} different abstractions for FS
- Most of them are too imprecise or costly
  - $P(\{x,y,z\}) = \{\cancel{\emptyset}, \cancel{\{x\}}, \cancel{\{y\}}, \cancel{\{z\}}, \{x,y\}, \cancel{\{y,z\}}, \cancel{\{x,z\}}, \{x,y,z\}\}$

Our Approaches

- Two approaches:
  - Using a meta pre-analysis [PLDI'14, TOPLAS'16]
  - Using machine learning [OOPSLA'15, SAS'16, APLAS,16]

Selective Context-Sensitivity (PLDI'14)

Example Program

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```

always holds

does not always hold

Context-Insensitivity

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```

Context-insensitive interval analysis cannot prove Q1

Context-Insensitivity

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```

[-∞,+∞]

Context-insensitive interval analysis cannot prove Q1

Context-Sensitivity: 3-CFA

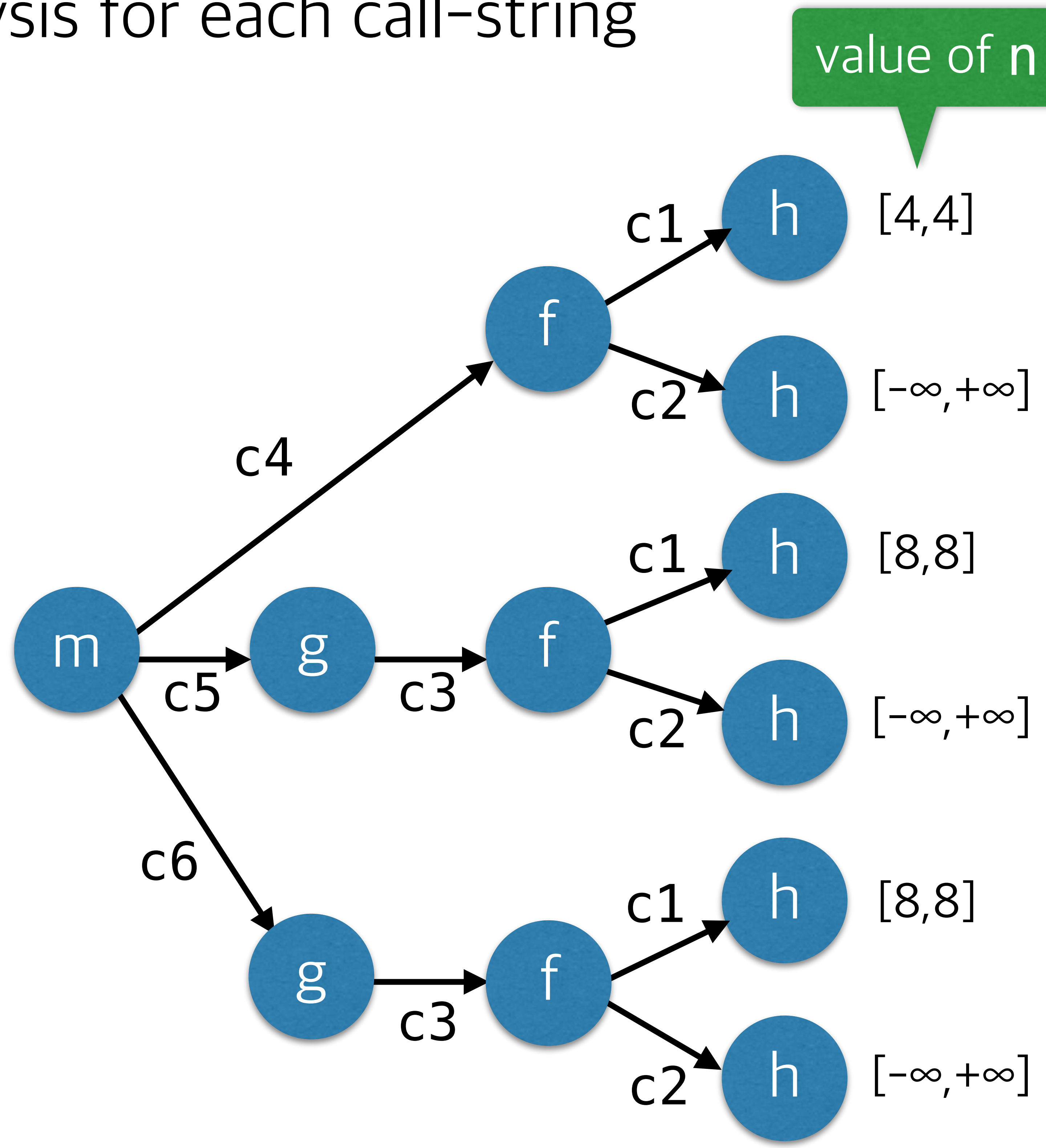
Separate analysis for each call-string

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



Context-Sensitivity: 3-CFA

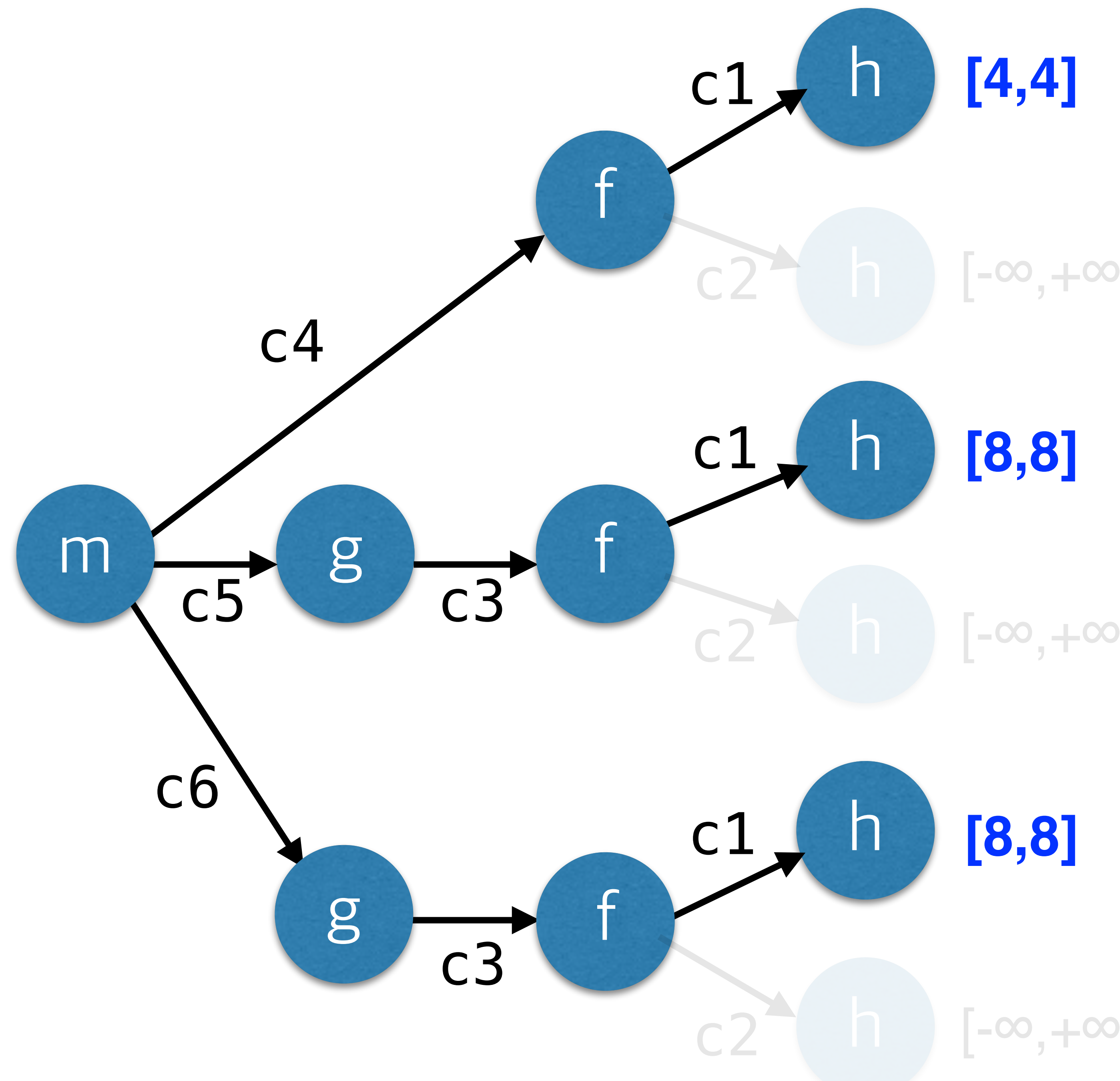
Separate analysis for each call-string

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



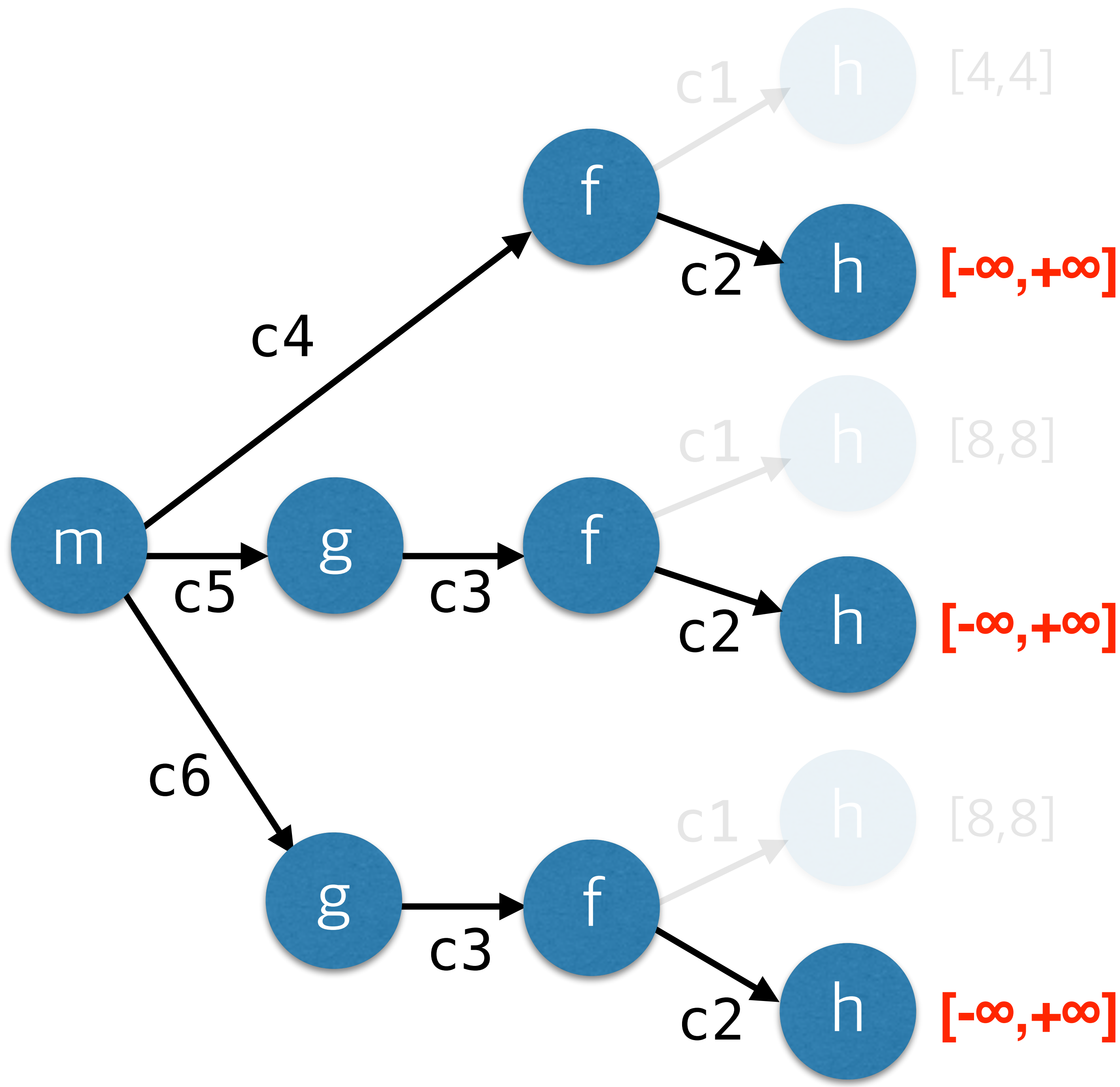
Problems of k-CFA

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



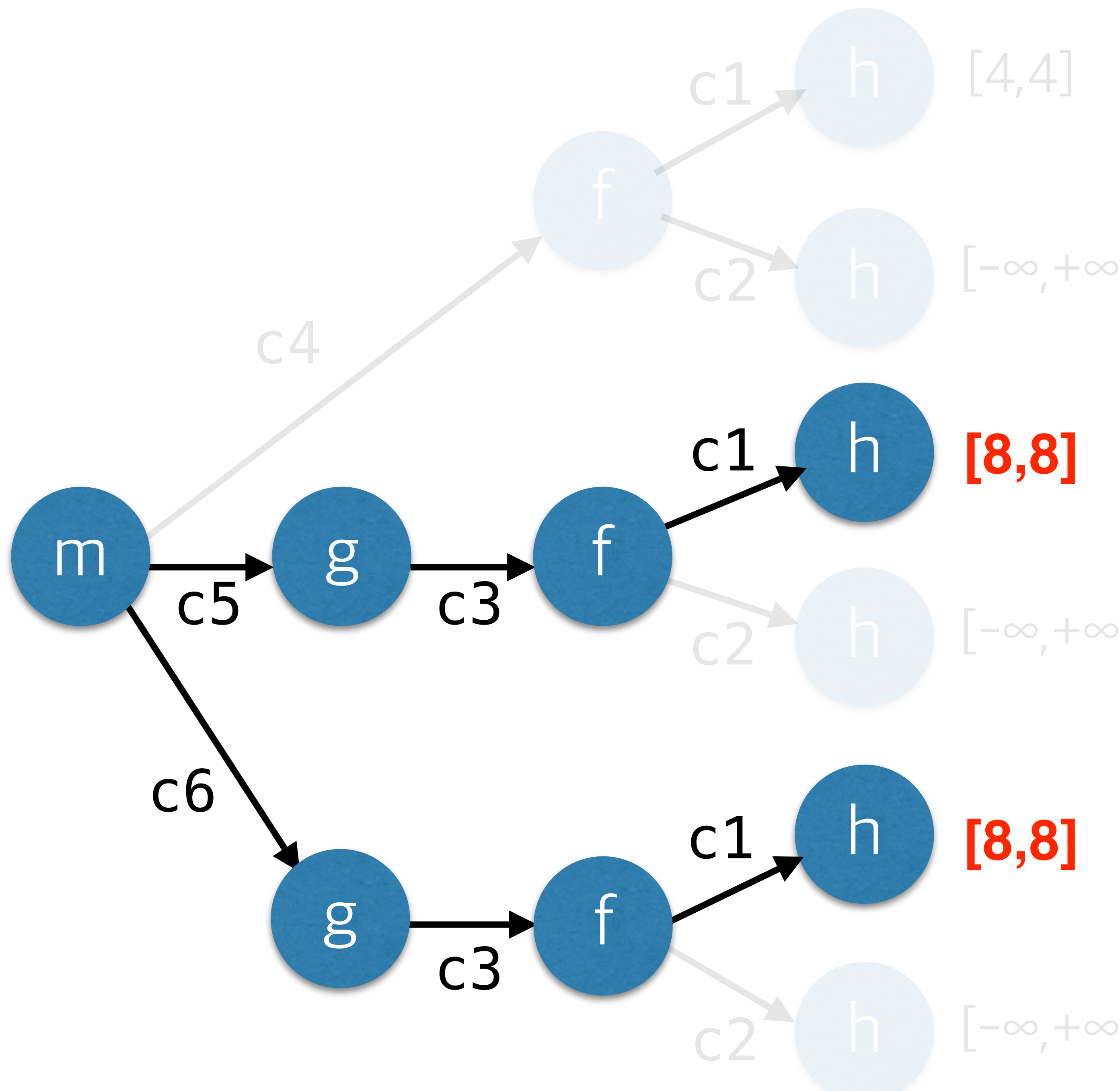
Problems of k-CFA

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



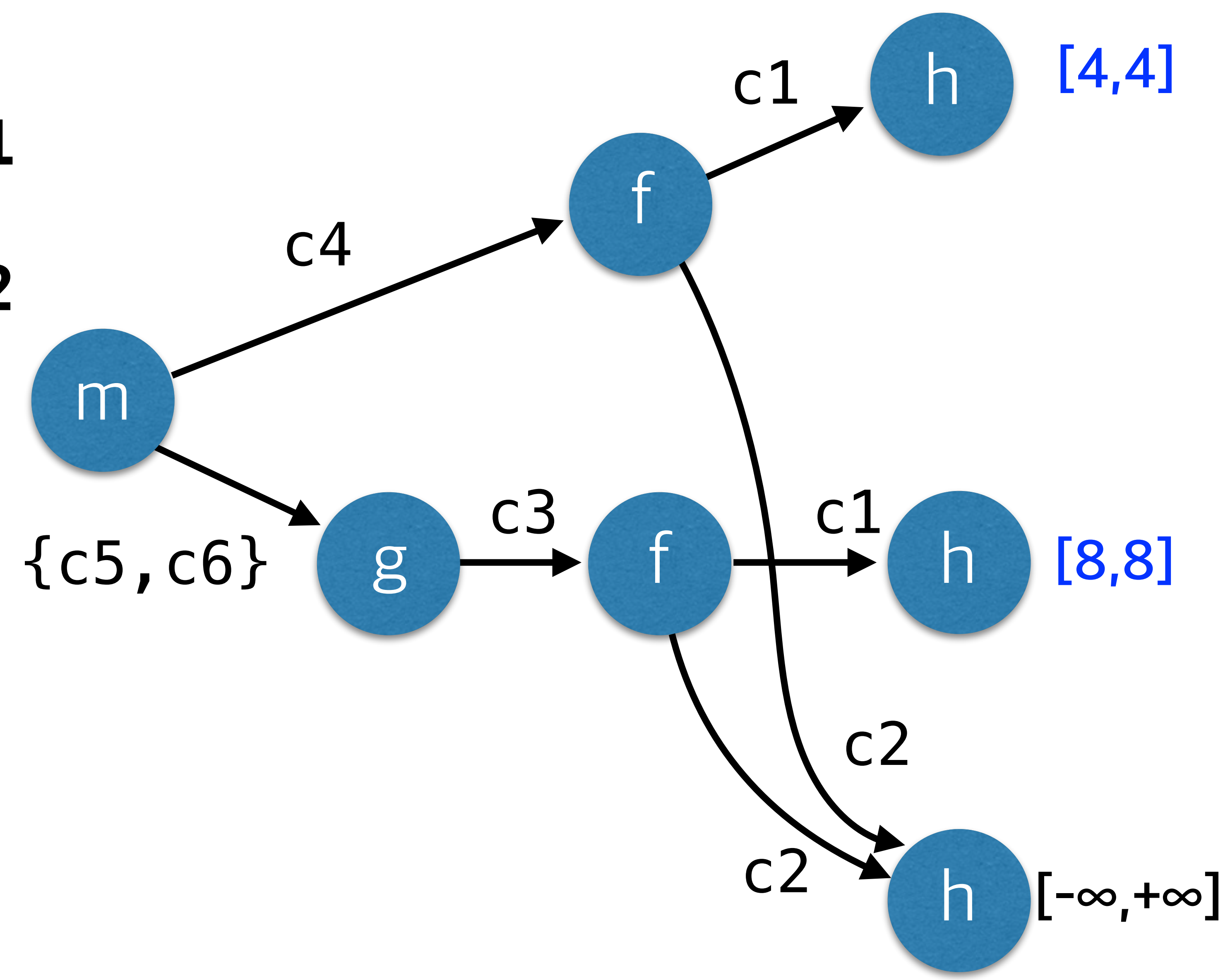
Our Selective Context-Sensitivity

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



Our Selective Context-Sensitivity

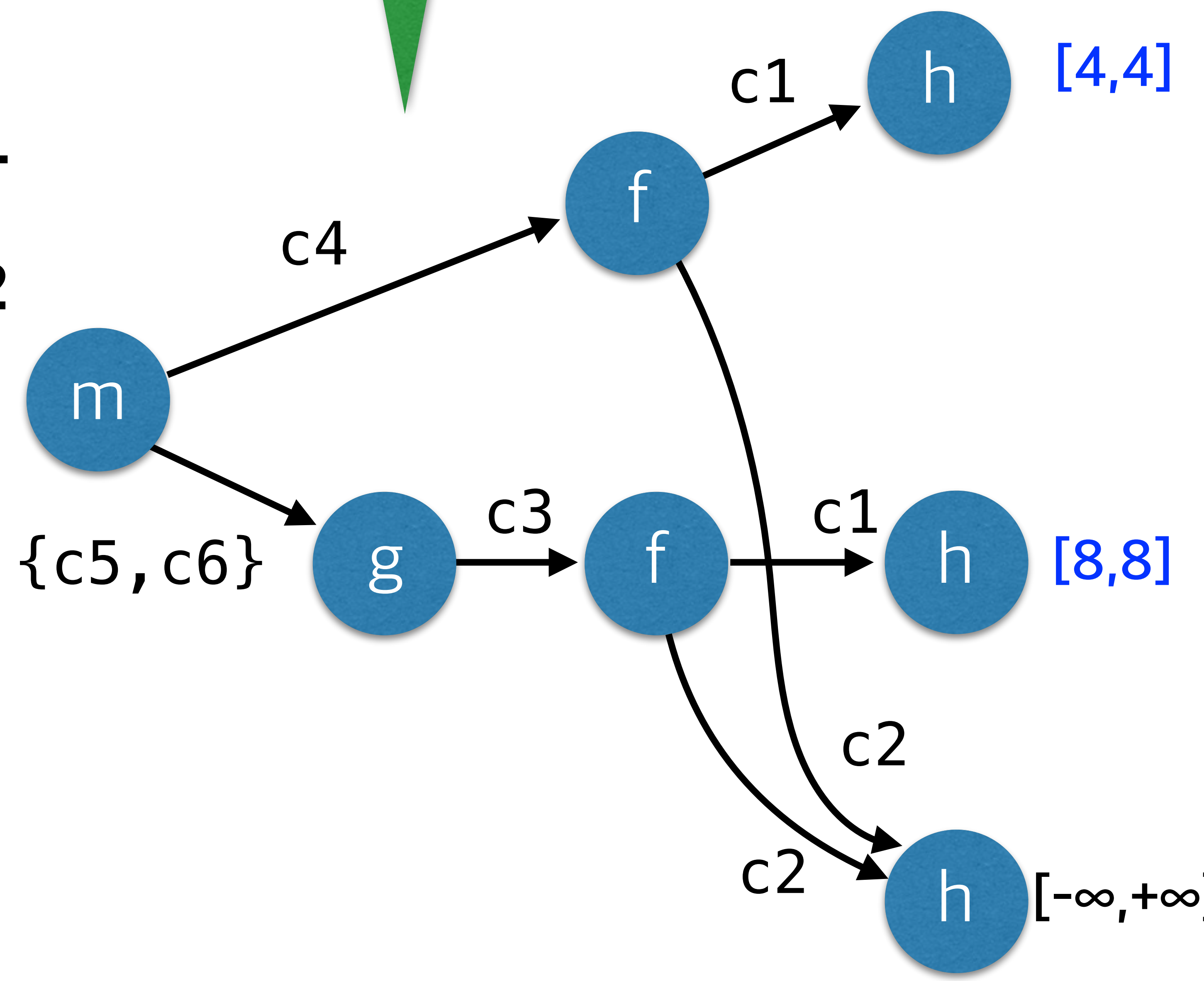
```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```

Challenge: How to infer this selective context-sensitivity?



Our Selective Context-Sensitivity

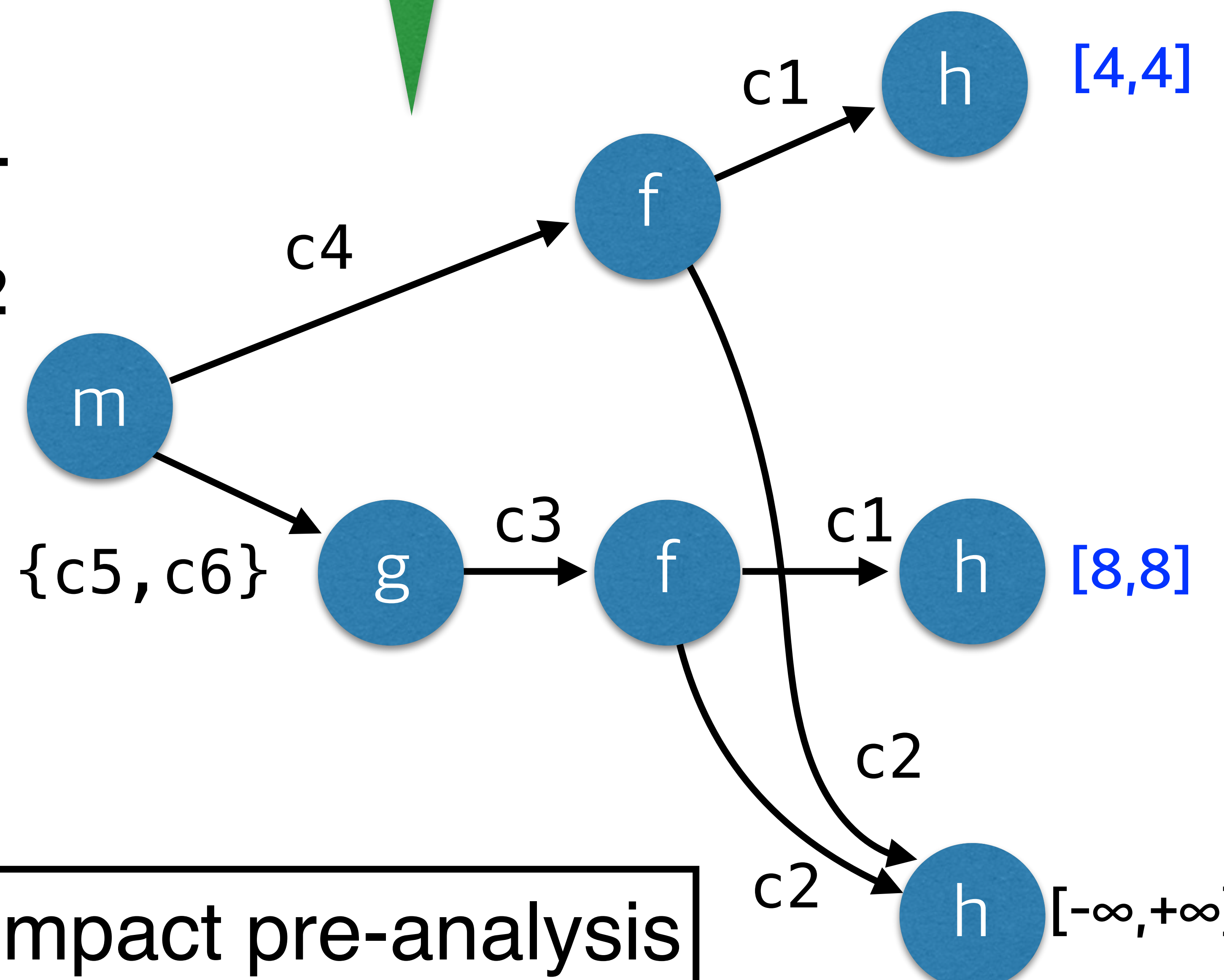
```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```

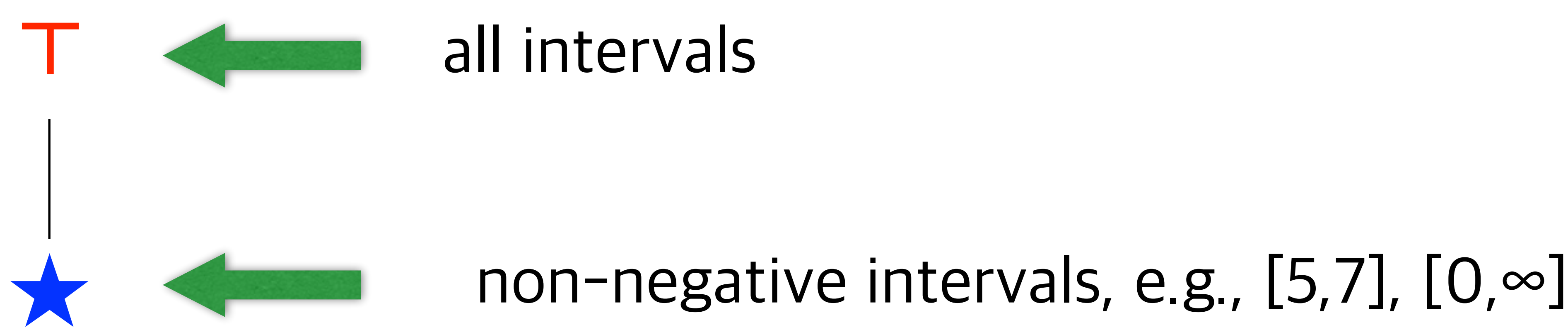
Challenge: How to infer this selective context-sensitivity?



Our solution: Impact pre-analysis

Impact Pre-Analysis

- Full context-sensitivity
- Approximate the interval domain



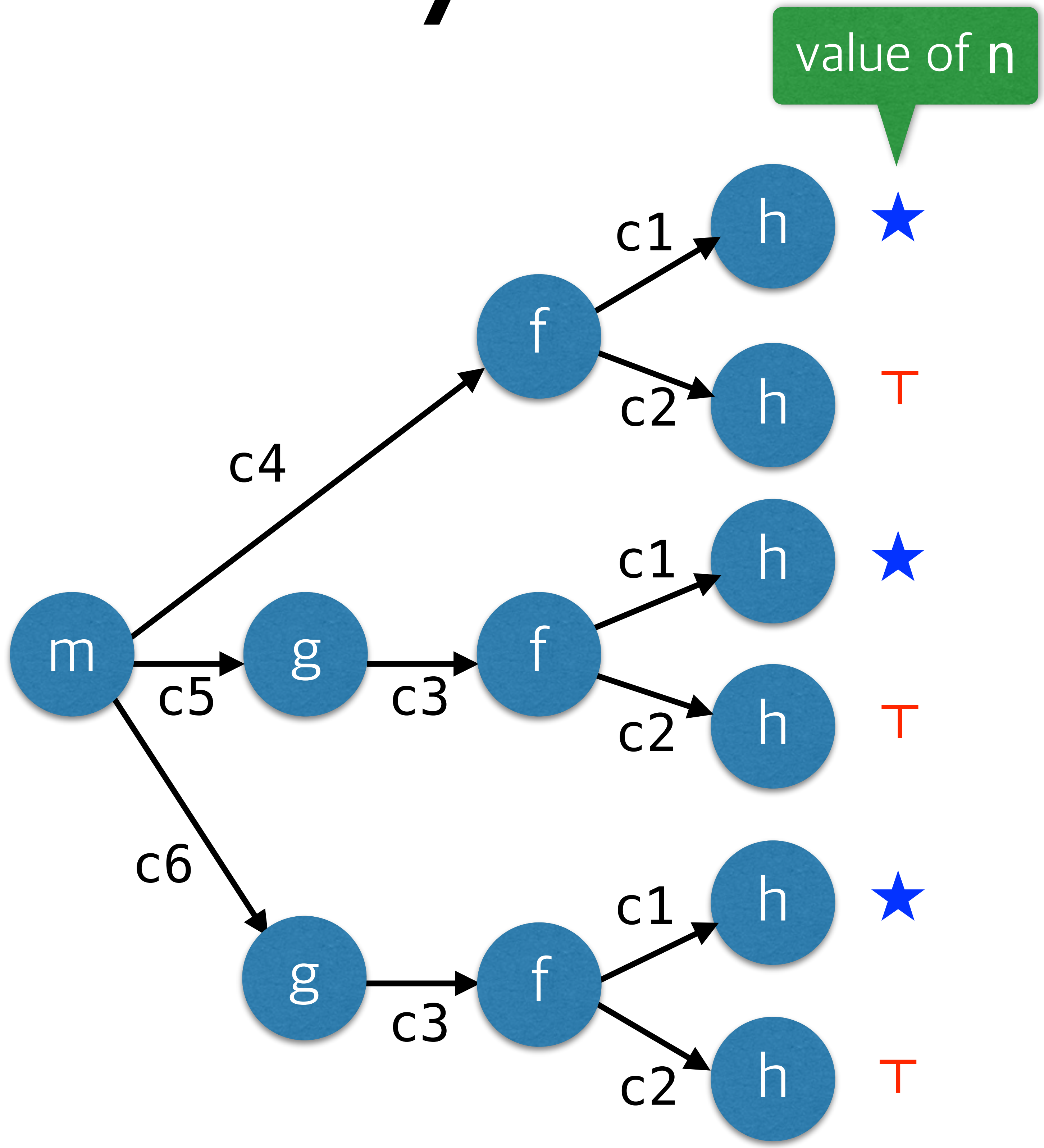
Impact Pre-Analysis

```
     int h(n) {ret n;}

     void f(a) {
c1:    x = h(a);
       assert(x > 1);  // Q1
c2:    y = h(input());
       assert(y > 1);  // Q2
     }

c3: void g() {f(8);}

     void m() {
c4:    f(4);
c5:    g();
c6:    g();
     }
```



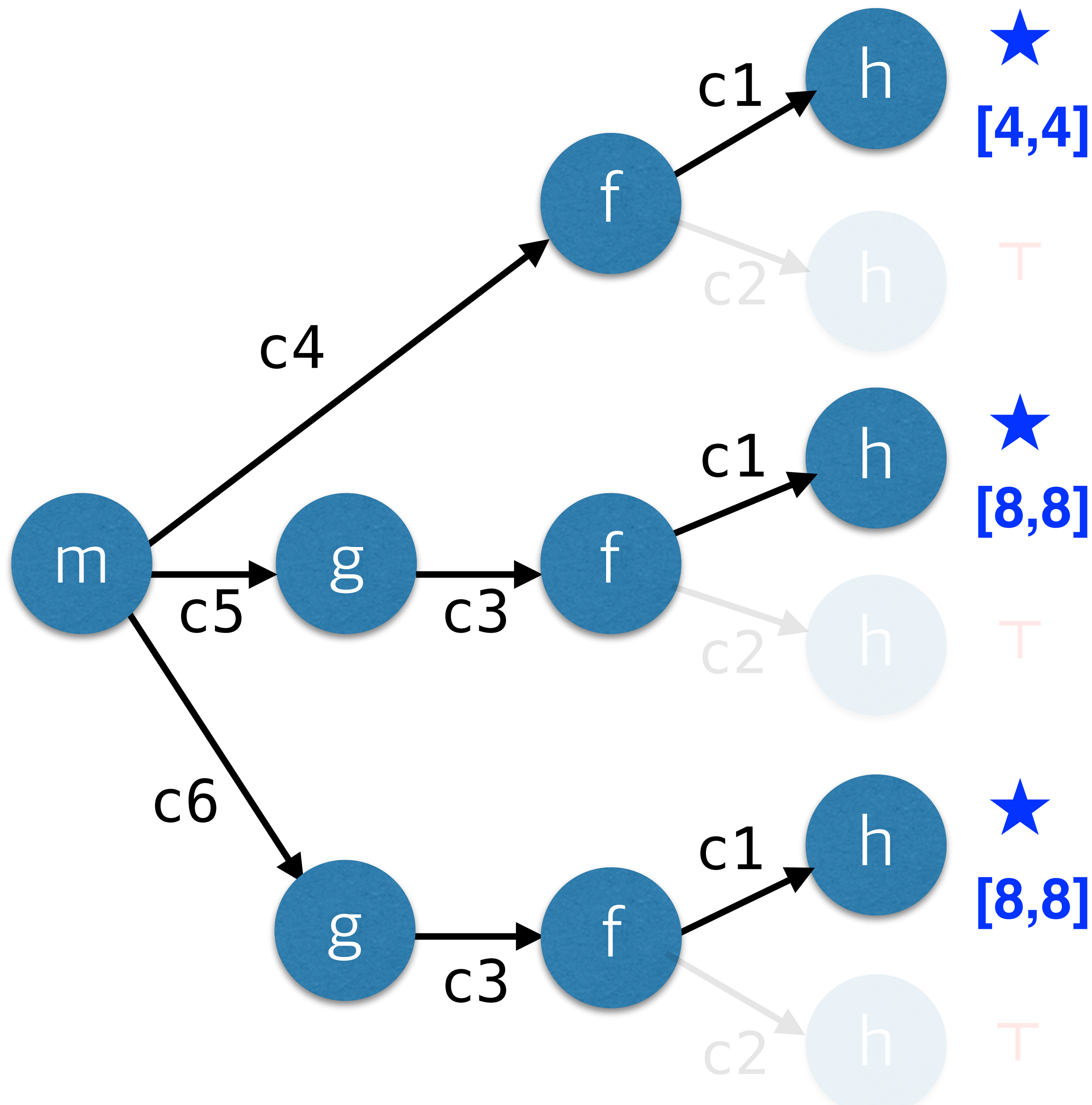
Impact Pre-Analysis

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



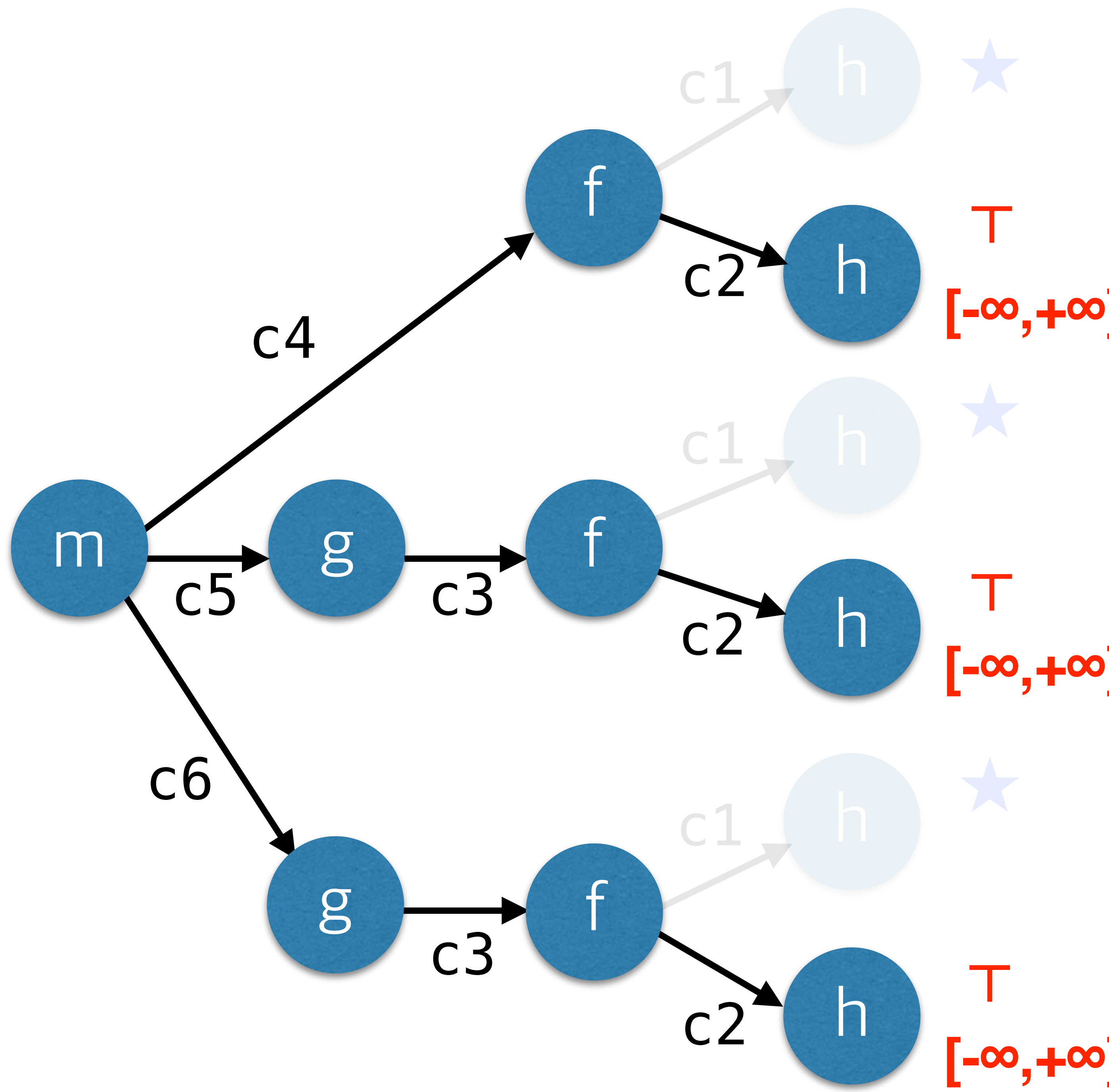
Impact Pre-Analysis

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



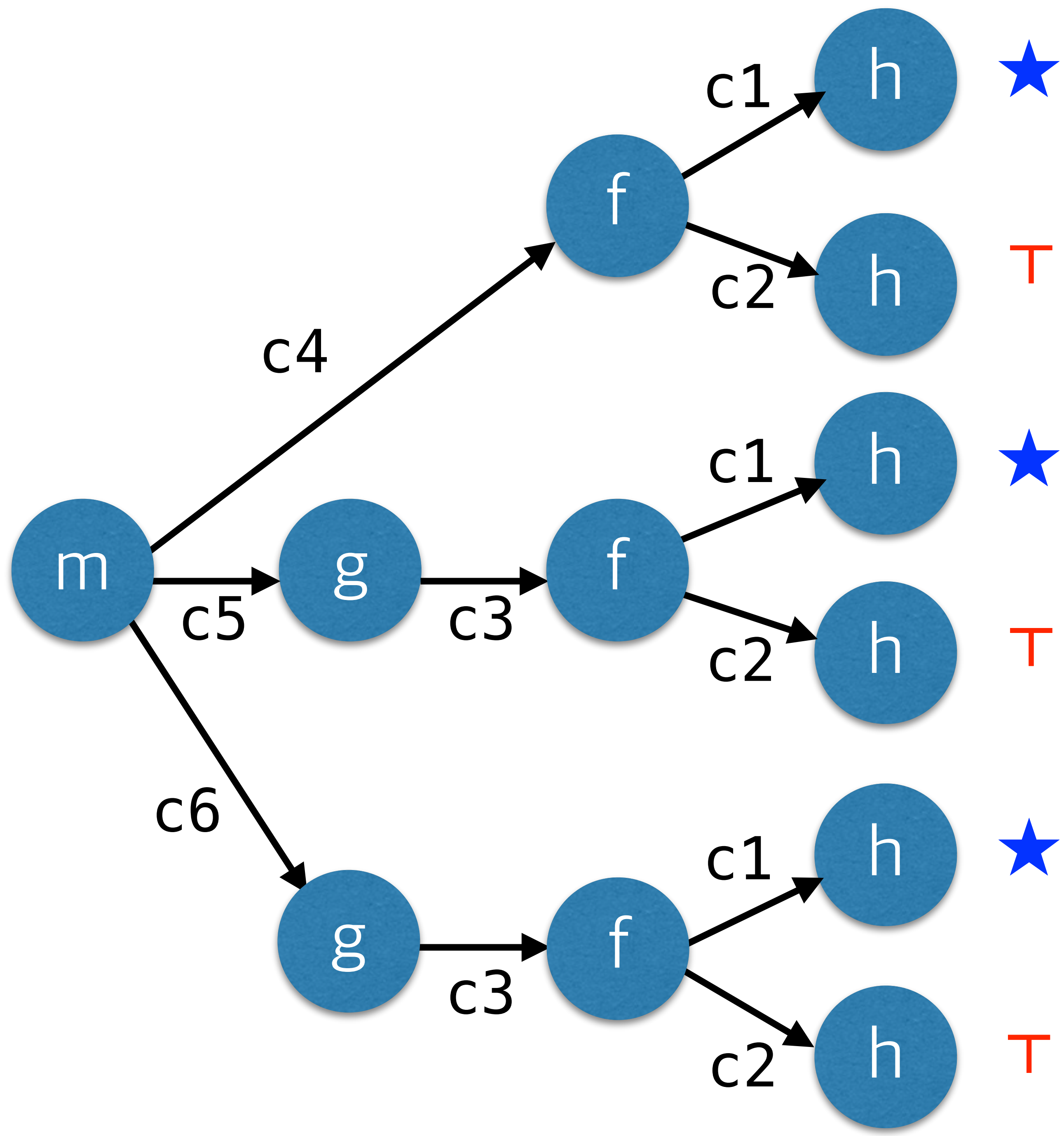
1. Collect queries whose expressions are assigned with ★

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



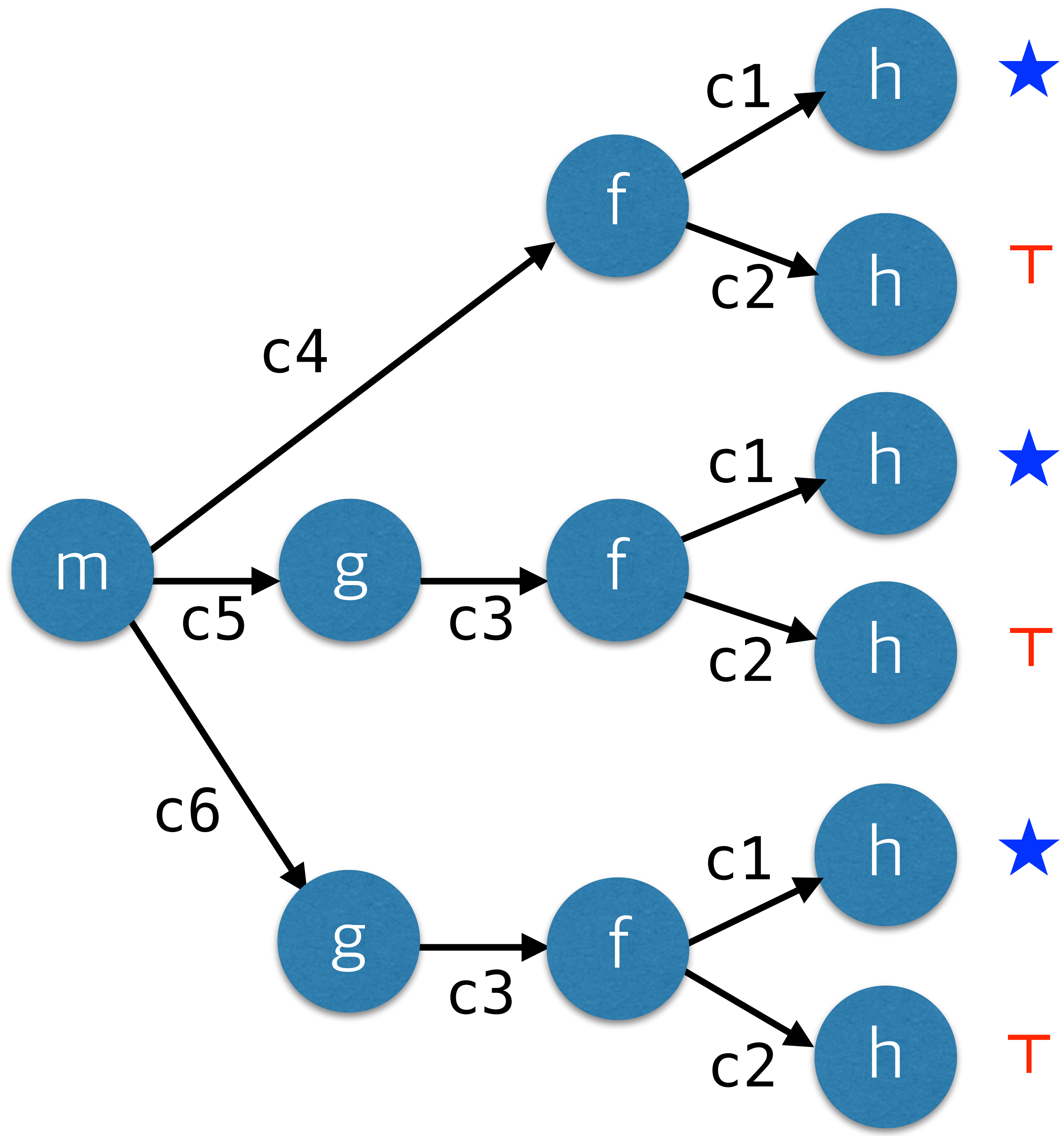
1. Collect queries whose expressions are assigned with ★

```
    int h(n) {ret n;}

    void f(a) {
c1: ★ x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



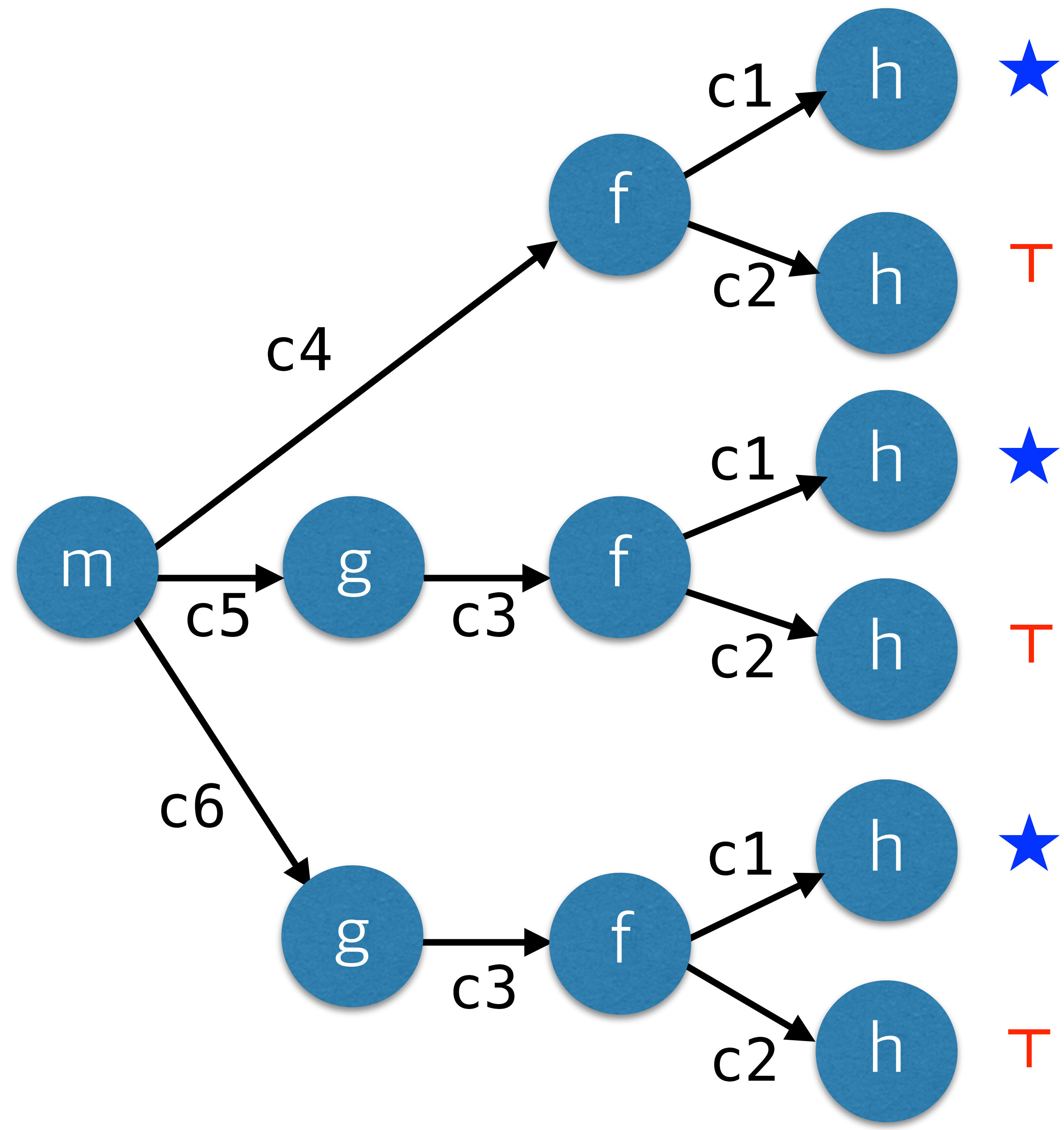
1. Collect queries whose expressions are assigned with ★

```
    int h(n) {ret n;}

    void f(a) {
c1: ★ x = h(a);
      assert(x > 1);  // Q1
c2: ⊤ y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



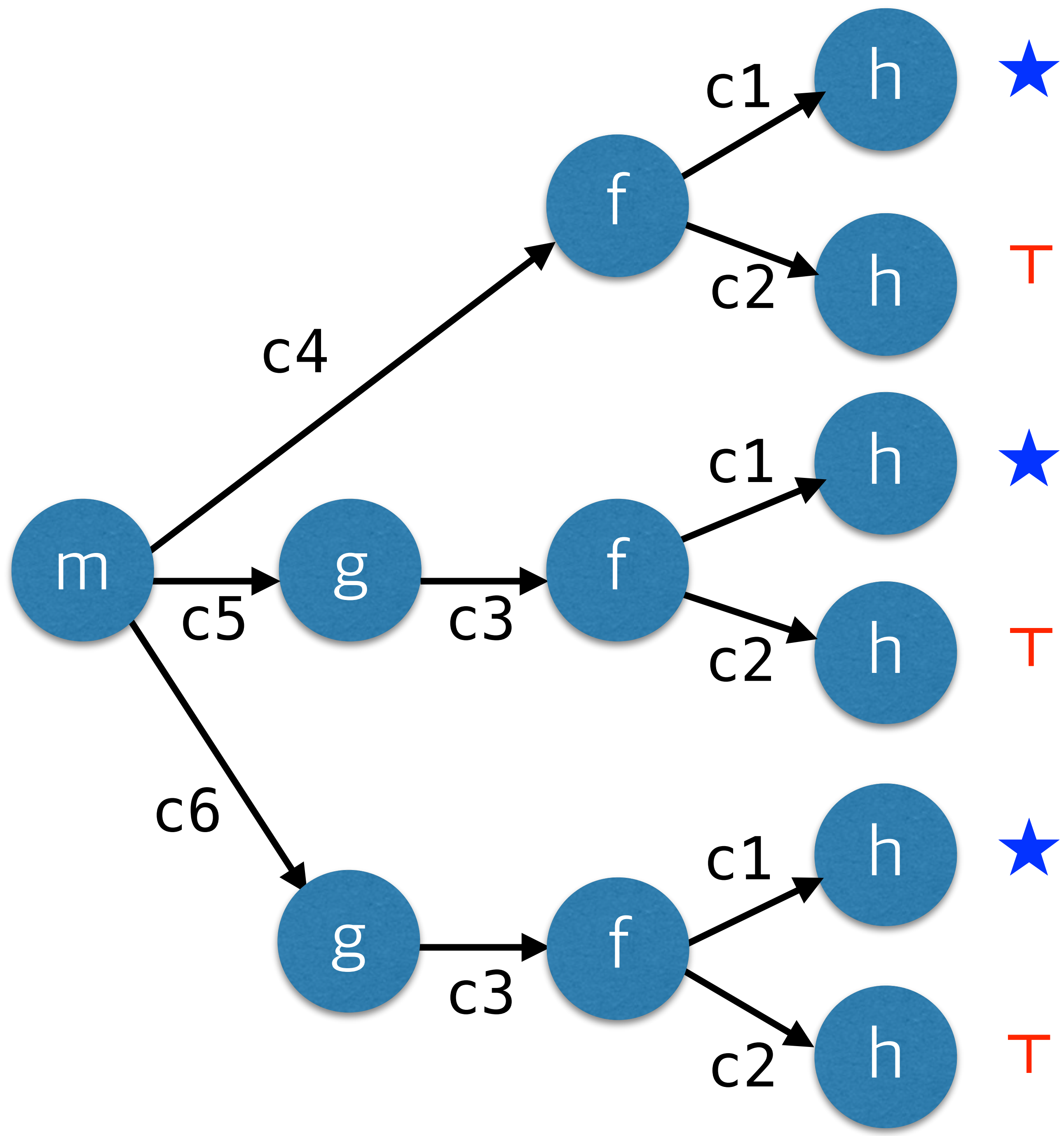
1. Collect queries whose expressions are assigned with ★

```
    int h(n) {ret n;}

    void f(a) {
c1: ★ x = h(a);
      assert(x > 1);   // Q1
c2: ⊤ y = h(input());
      assert(y > 1);   // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



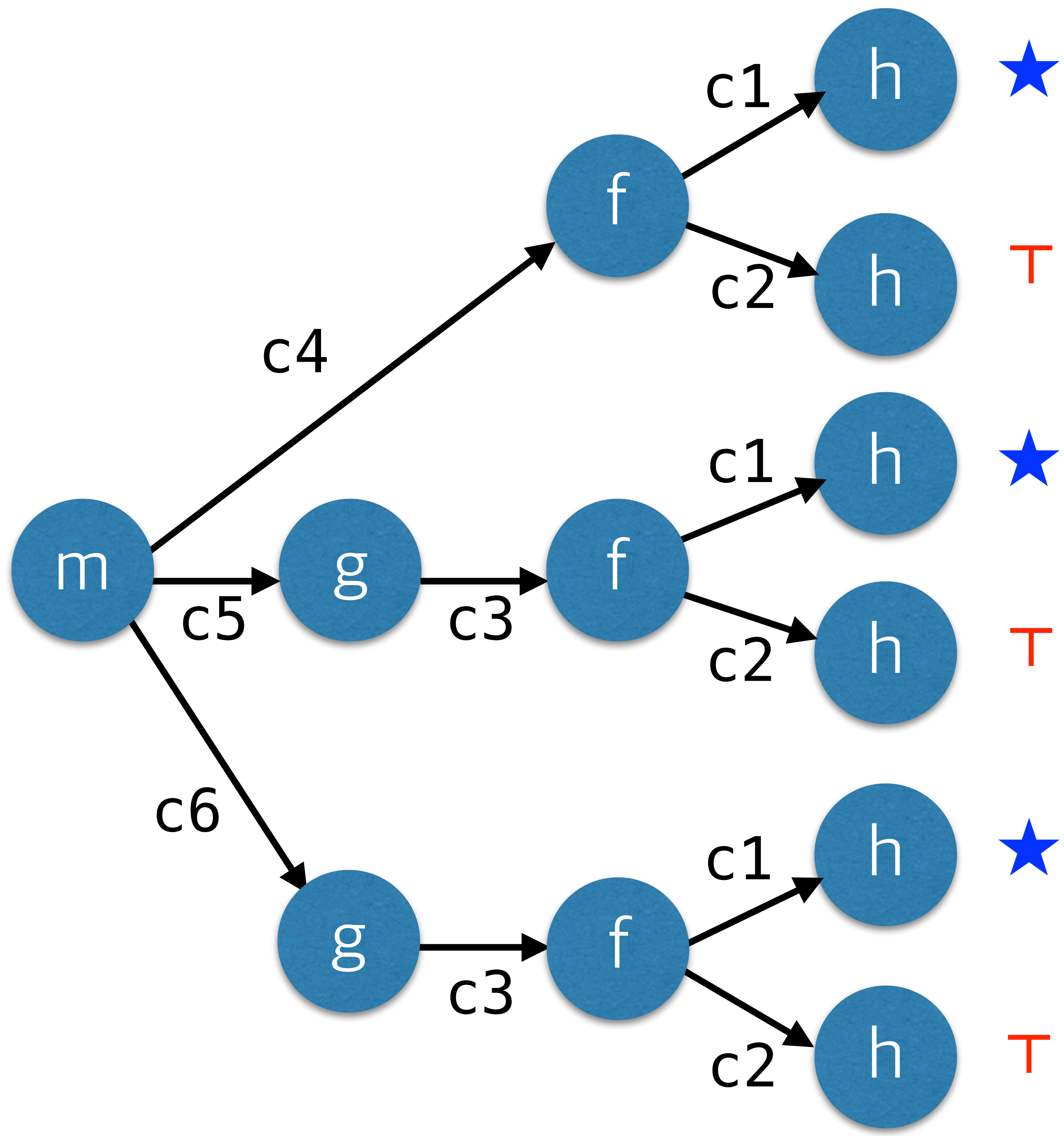
2. Find the program slice that contributes to the selected query

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



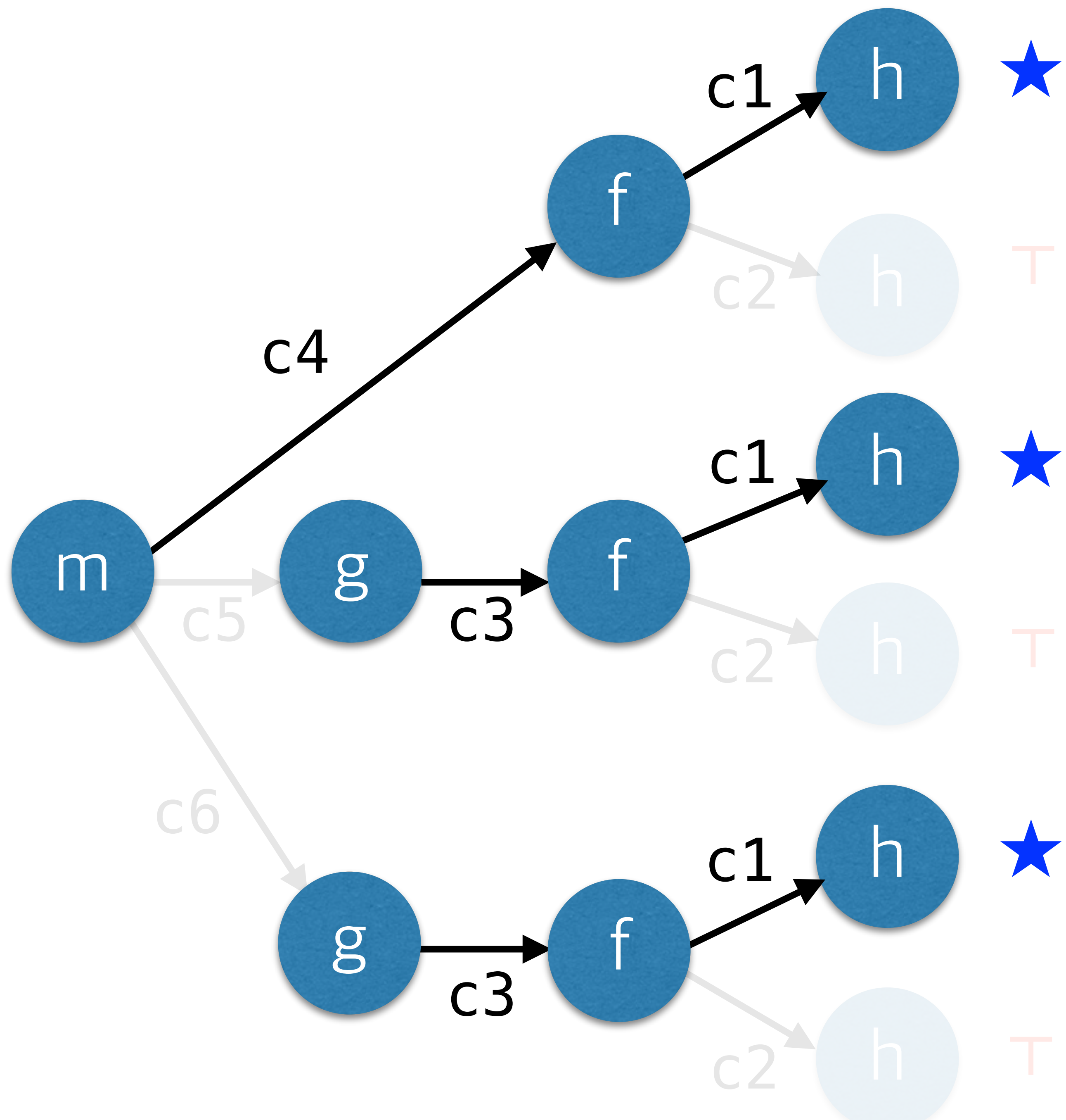
3. Collect contexts in the slice

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 1);  // Q1
c2:   y = h(input());
      assert(y > 1);  // Q2
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



=> Contexts for h: {c3 · c1, c4 · c1}

Partial Octagon Analysis

```
int a = b;
int c = input();
for (i = 0; i < b; i++) {
  assert (i < a);  // Q1
  assert (i < c);  // Q2
}
```

Partial Octagon Analysis

```
int a = b;
int c = input();
for (i = 0; i < b; i++) {
  assert (i < a);  // Q1
  assert (i < c);  // Q2
}
```

	a	b	c	i
a	0	0	∞	-1
b	0	0	∞	-1
c	∞	∞	0	∞
i	∞	∞	∞	0

non-selective analysis

Partial Octagon Analysis

i < b

```
int a = b;
int c = input();
for (i = 0; i < b; i++) {
  assert (i < a);  // Q1
  assert (i < c);  // Q2
}
```

	a	b	c	i
a	0	0	∞	-1
b	0	0	∞	-1
c	∞	∞	0	∞
i	∞	∞	∞	0

non-selective analysis

Partial Octagon Analysis

a = b

i < b

```
int a = b;
int c = input();
for (i = 0; i < b; i++) {
  assert (i < a);  // Q1
  assert (i < c);  // Q2
}
```

	a	b	c	i
a	0	0	∞	-1
b	0	0	∞	-1
c	∞	∞	0	∞
i	∞	∞	∞	0

non-selective analysis

Partial Octagon Analysis

a = b

i < b

```
int a = b;
int c = input();
for (i = 0; i < b; i++) {
  assert (i < a);  // Q1
  assert (i < c);  // Q2
}
```

	a	b	c	i
a	0	0	∞	-1
b	0	0	∞	-1
c	∞	∞	0	∞
i	∞	∞	∞	0

$i - a \leq -1$

non-selective analysis

Partial Octagon Analysis

a = b

i < b

```
int a = b;
int c = input();
for (i = 0; i < b; i++) {
  assert (i < a);  // Q1
  assert (i < c);  // Q2
}
```

	a	b	c	i
a	0	0	∞	$\boxed{-1}$
b	0	0	∞	-1
c	∞	∞	0	$\boxed{\infty}$
i	∞	∞	∞	0

$i - a \leq -1$

$i - c \leq \infty$

non-selective analysis

Partial Octagon Analysis

a = b

i < b

```
int a = b;
int c = input();
for (i = 0; i < b; i++) {
  assert (i < a);  // Q1
  assert (i < c);  // Q2
}
```

	a	b	c	i
a	0	0	∞	-1
b	0	0	∞	-1
c	∞	∞	0	∞
i	∞	∞	∞	0

$i - a \leq -1$

$i - c \leq \infty$

vs.

	a	b	i
a	0	0	-1
b	0	0	-1
i	∞	∞	0

non-selective analysis

our selective analysis

Impact Pre-Analysis

- Fully relational
- Approximated in other precision aspects

	a	b	c	i
a	0	0	∞	-1
b	0	0	∞	-1
c	∞	∞	0	∞
i	∞	∞	∞	0

octagon analysis

vs.

	a	b	c	i
a	★	★	$\top$	★
b	★	★	$\top$	★
c	$\top$	$\top$	★	$\top$
i	$\top$	$\top$	$\top$	★

impact pre-analysis

Selective Context-Sensitivity

		Context-Insensitve		Ours	
Pgm	LOC	#alarms	time(s)	#alarms	time(s)
spell	2K	58	0.6	30	0.9
bc	13K	606	14.0	483	16.2
tar	20K	940	42.1	799	47.2
less	23K	654	123.0	562	166.4
sed	27K	1,325	107.5	1,238	117.6
make	27K	1,500	88.4	1,028	106.2
grep	32K	735	12.1	653	15.9
wget	35K	1,307	69.0	942	82.1
a2ps	65K	3,682	118.1	2,121	177.7
bison	102K	1,894	136.3	1,742	173.4
TOTAL	346K	12,701	707.1	9,598	903.6

24.4%

Selective Context-Sensitivity

		Context-Insensitve		Ours	
Pgm	LOC	#alarms	time(s)	#alarms	time(s)
spell	2K	58	0.6	30	0.9
bc	13K	606	14.0	483	16.2
tar	20K	940	42.1	799	47.2
less	23K	654	123.0	562	166.4
sed	27K	1,325	107.5	1,238	117.6
make	27K	1,500	88.4	1,028	106.2
grep	32K	735	12.1	653	15.9
wget	35K	1,307	69.0	942	82.1
a2ps	65K	3,682	118.1	2,121	177.7
bison	102K	1,894	136.3	1,742	173.4
TOTAL	346K	12,701	707.1	9,598	903.6

27.8%

Selective Context-Sensitivity

		Context-Insensitve		Ours	
Pgm	LOC	#alarms	time(s)	#alarms	time(s)
spell	2K	58	0.6	30	0.9
bc	13K	606	14.0	483	16.2
tar	20K	940	42.1	799	47.2
less	23K	654	123.0	562	166.4
sed	27K	1,325	107.5	1,238	117.6
make	27K	1,500	88.4	1,028	106.2
grep	32K	735	12.1	653	15.9
wget	35K	1,307	69.0	942	82.1
a2ps	65K	3,682	118.1	2,121	177.7
bison	102K	1,894	136.3	1,742	173.4
TOTAL	346K	12,701	707.1	9,598	903.6

pre-analysis : 14.7%
main analysis: 13.1%

27.8%

k-CFA did not scale

- 2 or 3-CFA did not scale over 10KLoC
  - e.g., for spell (2KLoC):
    - 3-CFA reported 30 alarms in 11.9s
    - cf) ours: 30 alarms in 0.9s
- 1-CFA did not scale over 40KLoC

Selective Octagon Analysis

			Existing Approach [Miné06]		Ours	
Pgm	LOC	#queries	proven	time(s)	proven	time(s)
calc	298	10	2	0.3	10	0.2
spell	2,213	16	1	4.8	16	2.4
barcode	4,460	37	16	11.8	37	30.5
httptunnel	6,174	28	16	26.0	26	15.3
bc	13,093	10	2	247.1	9	117.3
tar	20,258	17	7	1043.2	17	661.8
less	23,822	13	0	3031.5	13	2849.4
a2ps	64,590	11	0	29473.3	11	2741.7
TOTAL	135,008	142	44	33840.3	139	6418.6

+95

Selective Octagon Analysis

			Existing Approach [Miné06]		Ours	
Pgm	LOC	#queries	proven	time(s)	proven	time(s)
calc	298	10	2	0.3	10	0.2
spell	2,213	16	1	4.8	16	2.4
barcode	4,460	37	16	11.8	37	30.5
httptunnel	6,174	28	16	26.0	26	15.3
bc	13,093	10	2	247.1	9	117.3
tar	20,258	17	7	1043.2	17	661.8
less	23,822	13	0	3031.5	13	2849.4
a2ps	64,590	11	0	29473.3	11	2741.7
TOTAL	**135,008**	**142**	**44**	**33840.3**	**139**	**6418.6**

reduce time by -81%

Framework

- For a range of static analyses,
  - how to design the impact pre-analysis
    - an efficient graph reachability-based algorithm
  - how to design selective context-sensitivity guide
    - soundness guarantee of the pre-analysis

Program Representation

- Control flow graph $(\mathbb{C}, \to, \mathbb{F}, \iota)$

$$\begin{array}{lllrlll} \mathbb{C} & = & \mathbb{C}_e & \text{(Entry Nodes)} & \uplus & \mathbb{C}_x & \text{(Exit Nodes)} \\ & \uplus & \mathbb{C}_c & \text{(Call Nodes)} & \uplus & \mathbb{C}_r & \text{(Return Nodes)} \\ & \uplus & \mathbb{C}_i & \text{(Internal Nodes)} & & & \end{array}$$

We associate a primitive command with each node c of our control flow graph, and denote it by $\mathsf{cmd}(c)$. For brevity, we consider simple primitive commands specified by the following grammar:

$$cmd \quad \to \quad skip \mid x := e$$

where e is an arithmetic expression: $e \to n \mid x \mid e + e \mid e - e$. We denote the set of all program variables by Var.

Program Representation

Each procedure $f \in \mathbb{F}$ has one entry node and one exit node. Given a node $c \in \mathbb{C}$, $\mathsf{fid}(c)$ denotes the procedure enclosing the node. Each call-site in the program is represented by a pair of call and return nodes. Given a return node $c \in \mathbb{C}_r$, we write $\mathsf{callof}(c)$ for the corresponding call node. We assume for simplicity that there are no indirect function calls such as calls via function pointers.

For simplicity, we handle parameter passing and return values of procedures via simple syntactic encoding. Recall that we represent a call statement $x := f_p(e)$ (where p is a formal parameter of procedure f) with call and return nodes. In our program, the call node has command $p := e$, so that the actual parameter e is assigned to the formal parameter p. For return values, we assume that each procedure f has a variable r_f and the return value is assigned to r_f: that is, we represent return statement `return` e of procedure f by $r_f := e$. The return node has command $x := r_f$, so that the return value is assigned to the original return variable. We assume that there are no global variables in the program, all parameters and local variables of procedures are distinct, and there are no recursive procedures.

A Family of Static Analyses

1. A domain $\mathbb{S}$ of abstract states. We assume that this domain has a complete lattice structure:

   $$(\mathbb{S}, \sqsubseteq, \bot, \top, \sqcup, \sqcap).$$

2. An initial abstract state at the entry of the `main` procedure:

   $$s_I \in \mathbb{S}.$$

3. An abstract semantics of every primitive command cmd:

   $$[\![cmd]\!] : \mathbb{S} \to \mathbb{S}.$$

   We require that semantic function $[\![cmd]\!]$ be monotone.

4. A context guide K that maps procedures to sets of calling contexts (sequences of call nodes):

   $$K \in \mathbb{F} \to \wp(\mathbb{C}_c^*)$$

   we write $\kappa \in K$ for $\kappa \in \bigcup_{f \in \mathbb{F}} K(f)$

A Family of Static Analyses

Context-enriched nodes & edges

$$\mathbb{C}_K = \{(c, \kappa) \mid c \in \mathbb{C} \ \wedge \ \kappa \in K(\mathsf{fid}(c))\}$$

Definition 1 ($\rightarrow_K$)**.** $(\rightarrow_K) \subseteq \mathbb{C}_K \times \mathbb{C}_K$ *is the context-enriched control flow relation:*

$(c, \kappa) \rightarrow_K (c', \kappa')$ *iff*

$$\begin{cases} c \rightarrow c' \ \wedge \ \kappa' = \kappa & (c' \notin \mathbb{C}_e \uplus \mathbb{C}_r) \\ c \rightarrow c' \ \wedge \ \kappa' = c ::_K \kappa & (c \in \mathbb{C}_c \ \wedge \ c' \in \mathbb{C}_e) \\ c \rightarrow c' \ \wedge \ \kappa = \mathsf{callof}(c') ::_K \kappa' & (c \in \mathbb{C}_x \ \wedge \ c' \in \mathbb{C}_r) \end{cases}$$

where $(::_K) \in \mathbb{C}_c \times \mathbb{C}_c^* \rightarrow \mathbb{C}_c^*$ *updates contexts according to* K*:*

$$c ::_K \kappa = \begin{cases} c \cdot \kappa & (c \cdot \kappa \in K) \\ \epsilon & \textit{otherwise} \end{cases}$$

Example

```
1   char* xmalloc (int n) { return malloc(n); }
2
3   void multi_glob (int size) {
4     p = xmalloc (size);
5     assert (sizeof(p) > 1);     // Query 1
6     q = xmalloc (input());
7     assert (sizeof(q) > 1);     // Query 2
8   }
9
10  void f (int x) { multi_glob (x); }
11  void g ()      { multi_glob (4); }
12
13  int main() {
14    f (8);
15    f (16);
16    g ();
17    g ();
18  }
```

context-insensitivity

$$K = \lambda f.\{\epsilon\}$$

full context-sensitivity

$$K = \lambda f.\mathbb{C}_c^*$$

our selective context-sensitivity

$$\begin{aligned} \mathit{main} &\mapsto \{\epsilon\} \\ f &\mapsto \{14, 15\} \\ g &\mapsto \{\epsilon\} \\ \mathit{multi\_glob} &\mapsto \{10 \cdot 14, 10 \cdot 15, 11\} \\ \mathit{xmalloc} &\mapsto \{4 \cdot 10 \cdot 14, 4 \cdot 10 \cdot 15, 4 \cdot 11, \epsilon\} \end{aligned}$$

A Family of Static Analyses

Abstract domain & semantic function

$$\mathbb{D} = (\mathbb{C}_K \to \mathbb{S})$$

$$F(X)(c,\kappa) = [\![\mathsf{cmd}(c)]\!]\Big(\bigsqcup_{(c_0,\kappa_0)\to_K(c,\kappa)} X(c_0,\kappa_0)\Big)$$

The analysis computes the least X such that

$$s_I \sqsubseteq X(\iota,\epsilon) \;\wedge\; \forall(c,\kappa)\in\mathbb{C}_K.\; F(X)(c,\kappa) \sqsubseteq X(c,\kappa)$$

Some analyses compute some solution using widening:

$$\nabla : \mathbb{D}\times\mathbb{D}\to\mathbb{D}.$$

Example 2 (Interval Analysis)**.** *The interval analysis is a standard example that uses a widening operator. Let* $\mathbb{I}$ *be the domain of intervals:* $\mathbb{I} = \{[l, u] \mid l, u \in \mathbb{Z} \cup \{-\infty, +\infty\} \wedge l \leq u\}$. *Using this domain, we specify the rest of the analysis:*

1. *The abstract states are* $\perp$ *or functions from program variables to their interval values:* $\mathbb{S} = \{\perp\} \cup (\mathsf{Var} \to \mathbb{I})$
2. *The initial abstract state is:* $s_I(x) = [-\infty, +\infty]$.
3. *The abstract semantics of primitive commands is:*

$$[\![skip]\!](s) = s, \quad [\![x := e]\!](s) = \begin{cases} s[x \mapsto [\![e]\!](s)] & (s \neq \perp) \\ \perp & (s = \perp) \end{cases}$$

where $[\![e]\!]$ *is the abstract evaluation of the expression* e*:*

$$\begin{array}{ll} [\![n]\!](s) = [n, n], & [\![e_1 + e_2]\!](s) = [\![e_1]\!](s) + [\![e_2]\!](s) \\ [\![x]\!](s) = s(x), & [\![e_1 - e_2]\!](s) = [\![e_1]\!](s) - [\![e_2]\!](s) \end{array}$$

4. *The last component of the analysis is a widening operator, which is defined as a pointwise lifting of the following widening operators* $\nabla_I : \mathbb{I} \times \mathbb{I} \to \mathbb{I}$ *for intervals:*

$$\begin{array}{rcl} [l, u] \nabla_I [l', u'] & = & [\mathrm{ite}(l' < l, \mathrm{ite}(l' < 0, -\infty, 0), l), \\ & & \mathrm{ite}(u' > u, +\infty, u)] \end{array}$$

where $\mathrm{ite}(p, a, b)$ *evaluates to* a *if* p *is true and* b *otherwise. The above widening operator uses* 0 *as a threshold, which is useful when proving the absence of buffer overruns.*

Queries

Queries Queries are triples in $\mathcal{Q} \subseteq \mathbb{C} \times \mathbb{S} \times \mathsf{Var}$, and they are given as input to our static analysis. A query (c, s, x) represents an assertion that every reachable concrete state at node c is over-approximated by the abstract state s. The last component x describes that the query is concerned with the value of the variable x. For instance, in the interval analysis, a typical query is

$$(c, \quad \lambda y.\, \text{if } (y = x) \text{ then } [0, \infty] \text{ else } \top, \quad x)$$

for some variable x. It asserts that at program node c, the variable x should always have a non-negative value. Proving the queries or identifying those that are likely to be violated is the goal of the analysis.

Impact Pre-Analysis

$$(\mathbb{S}^\sharp, \;\; s_I^\sharp \in \mathbb{S}^\sharp, \;\; [\![-]\!]^\sharp : \mathbb{S}^\sharp \to \mathbb{S}^\sharp, \;\; K_\infty)$$

$$\mathbb{D}^\sharp \;=\; \mathbb{C}_{K_\infty} \;\to\; \mathbb{S}^\sharp$$

$$F^\sharp(X)(c,\kappa) = [\![\mathsf{cmd}(c)]\!]^\sharp(\bigsqcup_{(c_0,\kappa_0)\to_{K_\infty}(c,\kappa)} X(c_0,\kappa_0)).$$

It computes the least X satisfying

$$s_I^\sharp \sqsubseteq X(\iota,\epsilon) \;\wedge\; \forall(c,\kappa) \in \mathbb{C}_K.\; F^\sharp(X)(c,\kappa) \sqsubseteq X(c,\kappa) \qquad (6)$$

Soundness Conditions

1. $\gamma : \mathbb{S}^\sharp \to \wp(\mathbb{S})$.
2. $s_I \in \gamma(s_I^\sharp)$.
3. $\forall s \in \mathbb{S}, s^\sharp \in \mathbb{S}^\sharp.\ s \in \gamma(s^\sharp) \implies [\![cmd]\!](s) \in \gamma([\![cmd]\!]^\sharp(s^\sharp))$.
4. $\forall X, Y \in \mathbb{D}.\ \forall X^\sharp, Y^\sharp \in \mathbb{D}^\sharp.\ (X \in \gamma(X^\sharp) \wedge Y \in \gamma(Y^\sharp))$
$$\implies X \bigtriangledown Y \in \gamma(X^\sharp \sqcup Y^\sharp).$$

Lemma 1. *Let $M \in \mathbb{D}$ be the main analysis result, i.e., a solution of* (5) *under full context-sensitivity ($K = K_\infty$). Let $P \in \mathbb{D}^\sharp$ be the pre-analysis result, i.e., the least solution of* (6)*. Then,* $\forall c \in \mathbb{C}, \kappa \in \mathbb{C}_c^*.\ M(c, \kappa) \in \gamma(P(c, \kappa))$.

Efficiency Conditions

1. The abstract states are $\bot$ or functions from program variables to abstract values: $\mathbb{S}^\sharp = \{\bot\} \cup (\mathsf{Var} \to \mathbb{V})$, where $\mathbb{V}$ is a finite complete lattice $(\mathbb{V}, \sqsubseteq_v, \bot_v, \top_v, \sqcup_v, \sqcap_v)$. An initial abstract state is $s^\sharp_I = \lambda x.\, \top_v$.

2. The abstract semantics of primitive commands has a simple form involving only join operation and constant abstract value, which is defined as follows:

   $$[\![skip]\!]^\sharp(s) = s, \quad [\![x := e]\!]^\sharp(s) = \begin{cases} s[x \mapsto [\![e]\!]^\sharp(s)] & (s \neq \bot) \\ \bot & (s = \bot) \end{cases}$$

   where $[\![e]\!]^\sharp$ has the following form: for every $s \neq \bot$,

   $$[\![e]\!]^\sharp(s) = s(x_1) \sqcup \ldots \sqcup s(x_n) \sqcup v$$

   for some variables $x_1, \ldots, x_n$ and an abstract value $v \in \mathbb{V}$, all of which are fixed for the given e. We denote these variables and the value by

   $$\mathsf{var}(e) = \{x_1, \ldots, x_n\}, \qquad \mathsf{const}(e) = v.$$

Example

1. *Let* $\mathbb{V} = \{\bot_v, \bigstar, \top_v\}$ *be a lattice such that* $\bot_v \sqsubseteq_v \bigstar \sqsubseteq_v \top_v$. *Define the function* $\gamma_v : \{\bot_v, \bigstar, \top_v\} \to \wp(\mathbb{I})$ *as follows:*

$$\gamma_v(\top_v) = \mathbb{I}, \quad \gamma_v(\bigstar) = \{[a,b] \in \mathbb{I} \mid 0 \le a\}, \quad \gamma_v(\bot_v) = \emptyset$$

2. *The domain of abstract states is defined as* $\mathbb{S}^\sharp = \{\bot\} \cup (\mathsf{Var} \to \mathbb{V})$. *The meaning of abstract states in* $\mathbb{S}^\sharp$ *is given by* γ *such that* $\gamma(\bot) = \{\bot\}$ *and, for* $s^\sharp \neq \bot$,

$$\gamma(s^\sharp) = \{s \in \mathbb{S} \mid s = \bot \ \vee \ \forall x \in \mathsf{Var}.\ s(x) \in \gamma_v(s^\sharp(x))\}.$$

3. *Initial abstract state:* $s^\sharp_I = \top = \lambda x.\top_v$.
4. *Abstract evaluation* $[\![e]\!]^\sharp$ *of expression* e*: for every* $s \neq \bot$,

$$\begin{array}{ll} [\![n]\!](s) = \mathrm{ite}(n \ge 0, \bigstar, \top_v), & [\![e_1 + e_2]\!](s) = [\![e_1]\!](s) \sqcup_v [\![e_2]\!](s) \\ [\![x]\!](s) = s(x), & [\![e_1 - e_2]\!](s) = \top_v \end{array}$$

Reachability-based Algorithm

- Value-flow graph: $\Theta = \mathbb{C} \times \mathsf{Var}, \qquad (\hookrightarrow) \subseteq \Theta \times \Theta$

Definition 2 $(\hookrightarrow)$**.** *The value-flow relation* $(\hookrightarrow) \subseteq (\mathbb{C} \times \mathsf{Var}) \times (\mathbb{C} \times \mathsf{Var})$ *links the vertices in* Θ *based on how values of variables flow to other variables in each primitive command:*

$$(c, x) \hookrightarrow (c', x') \;\; \textit{iff}$$
$$\begin{cases} c \to c' \;\wedge\; x = x' & (\mathsf{cmd}(c') = \mathit{skip}) \\ c \to c' \;\wedge\; x = x' & (\mathsf{cmd}(c') = y := e \;\wedge\; y \neq x') \\ c \to c' \;\wedge\; x \in \mathsf{var}(e) & (\mathsf{cmd}(c') = y := e \;\wedge\; y = x') \end{cases}$$

Reachability-based Algorithm

Definition 3 ($\hookrightarrow_K$)**.** *The context-enriched value-flow relation* $(\hookrightarrow_K) \subseteq (\mathbb{C}_K \times \mathsf{Var}) \times (\mathbb{C}_K \times \mathsf{Var})$ *links the vertices in* $\mathbb{C}_K \times \mathsf{Var}$ *according to the specification below:*

$$((c,\kappa),x) \hookrightarrow_K ((c',\kappa'),x') \;\; \textit{iff}$$

$$\begin{cases} (c,\kappa) \to_K (c',\kappa') \;\wedge\; x = x' & (\mathsf{cmd}(c') = \mathit{skip}) \\ (c,\kappa) \to_K (c',\kappa') \;\wedge\; x = x' & (y \neq x') \\ (c,\kappa) \to_K (c',\kappa') \;\wedge\; x \in \mathsf{var}(e) & (y = x') \end{cases}$$

(where $\mathsf{cmd}(c')$ *in the last two cases is* $y := e$*)*

Reachability-based Algorithm

Definition 4 ($\hookrightarrow^\dagger_K$)**.** *The reachability relation* $(\hookrightarrow^\dagger_K) \subseteq \Theta \times \Theta$ *connects two vertices when one node can reach the other via an interprocedurally-valid path:*

$$\begin{array}{l}(c, x) \hookrightarrow^\dagger_K (c', x') \ \textit{iff} \\ \exists \kappa, \kappa'.\ (\iota, \epsilon) \rightarrow^*_K (c, \kappa) \ \wedge \ ((c, \kappa), x) \hookrightarrow^*_K ((c', \kappa'), x').\end{array}$$

We use the tabulation algorithm in [14] for computing $(\hookrightarrow^\dagger_K)$. While computing $(\hookrightarrow^\dagger_K)$, the algorithm also collects the set C of reachable nodes: $C = \{c \mid \exists \kappa.\, (\iota, \epsilon) \rightarrow^*_K (c, \kappa)\}$.

Reachability-based Algorithm

Third, our algorithm computes a set Θ_v of generators for each abstract value v in $\mathbb{V}$. Generators for v are vertices in Θ whose commands join v in their abstract semantics:

$$\begin{aligned}\Theta_v = \{(c, x) \mid \mathsf{cmd}(c) = x := e \wedge \mathsf{const}(e) = v\} \\ \cup\, (\textit{if}\ (v = \top_v)\ \textit{then}\ \{(\iota, x) \mid x \in \mathsf{Var}\}\ \textit{else}\ \{\})\end{aligned}$$

Definition 5 (PA_K)**.** $\mathsf{PA}_K \in \mathbb{C} \to \mathbb{S}^\sharp$ *is defined as follows:*

$$\begin{aligned}\mathsf{PA}_K(c) = &\ \textit{if}\ (c \notin C)\ \textit{then}\ \bot \\ &\ \textit{else}\ \lambda x. \bigsqcup \{v \in \mathbb{V} \mid \exists (c_0, x_0) \in \Theta_v . (c_0, x_0) \hookrightarrow^{\dagger}_{K} (c, x)\}.\end{aligned}$$

Then, PA_K is the solution of our pre-analysis:

Lemma 2. *Let X be the least solution satisfying* (6). *Then,* $\mathsf{PA}_K(c) = \bigsqcup_{\kappa \in \mathbb{C}^*} X(c, \kappa)$.

Query Selection

- select queries for which the analysis does not lose too much information

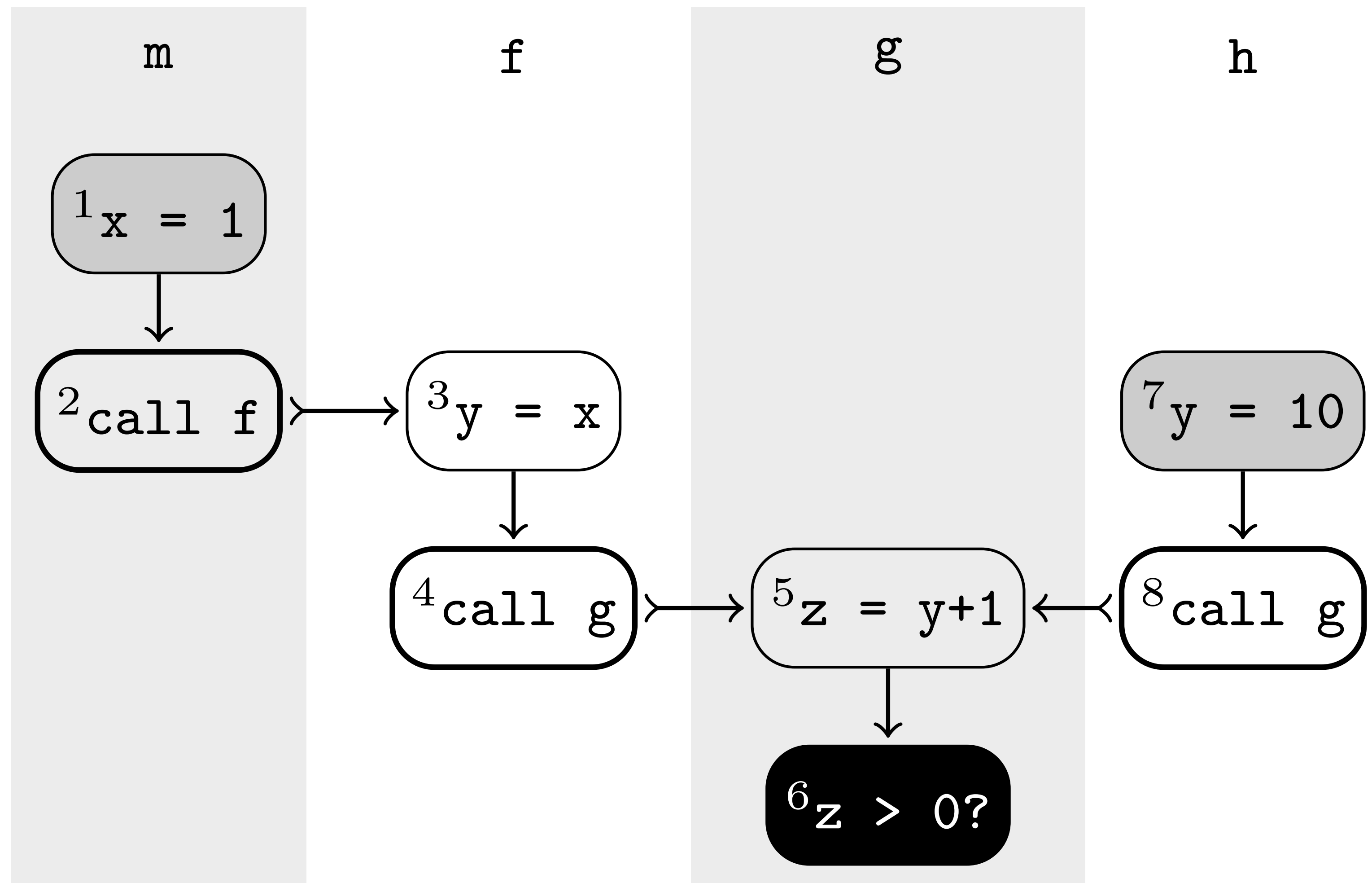
$$\begin{aligned}\mathcal{Q}^\sharp = \{(c,x) \in (\mathbb{C} \times \mathsf{Var}) \mid \exists s \in \mathbb{S}.\\ (c,s,x) \in \mathcal{Q} \;\wedge\; \forall s' \in \gamma(\mathsf{PA}_{K_\infty}(c)).\ s \sqcup s' \neq \top\}\end{aligned} \tag{7}$$

- in our case,

$$\mathsf{PA}_{K_\infty}(c)(x) \;\sqsubseteq\; \bigstar.$$

Building a Context Selector

CFG



Calling Contexts	κ_0	$2\cdot\kappa_0$	$\{4\cdot 2\cdot\kappa_0, 8\cdot\kappa_1\}$	κ_1

Context Selector $\quad \mathsf{K} = \{\mathtt{m} \mapsto \epsilon, \mathtt{f} \mapsto \{2, \epsilon\}, \mathtt{g} \mapsto \{4\cdot 2, 8\}, \mathtt{h} \mapsto \epsilon\}$

Theoretical Guarantee

Proposition 1 (Impact Realization)**.** *Let* $\mathsf{PA}_{K_\infty} \in \mathbb{C} \to \mathbb{S}^\sharp$ *be the result of the impact pre-analysis (Definition 5). Let* $q \in \mathcal{Q}^\sharp$ *be a selected query* (7)*. Let* K *be the context selector for* q *(Definition 10) defined using the pre-analysis result* PA_{K_∞}*. Let* $\mathsf{MA}_\mathsf{K} \in \mathbb{C}_K \to \mathbb{S}$ *be the main analysis result with the context selector* K*. Then, the selective main analysis is at least as precise as the fully context-sensitive pre-analysis for the selected query* q*:*

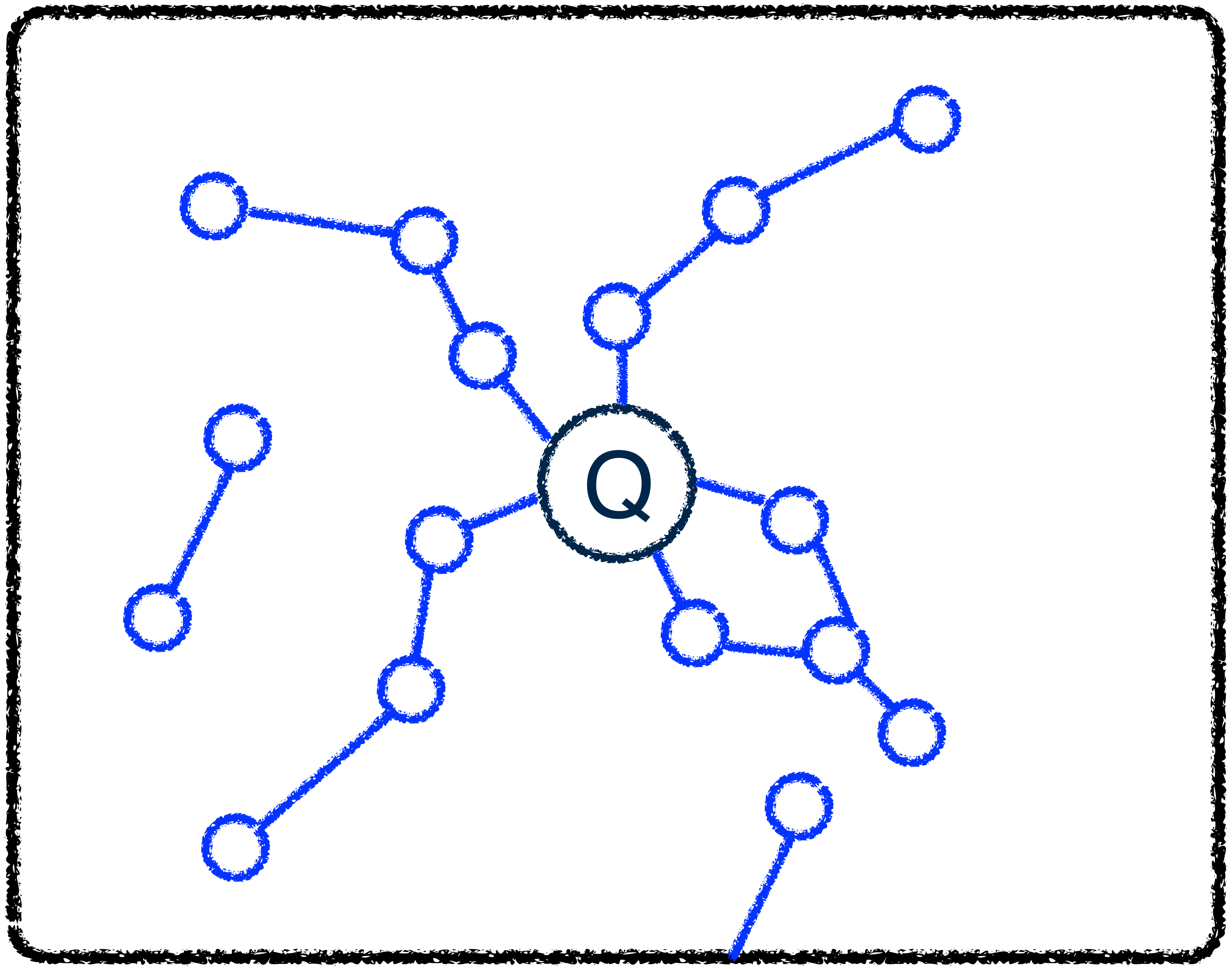
$$\mathsf{MA}_\mathsf{K} \sqsubseteq_q \mathsf{PA}_{K_\infty}$$

where $\mathsf{MA}_\mathsf{K} \sqsubseteq_q \mathsf{PA}_{K_\infty}$ *iff* $(q \overset{let}{=} (c, x))$

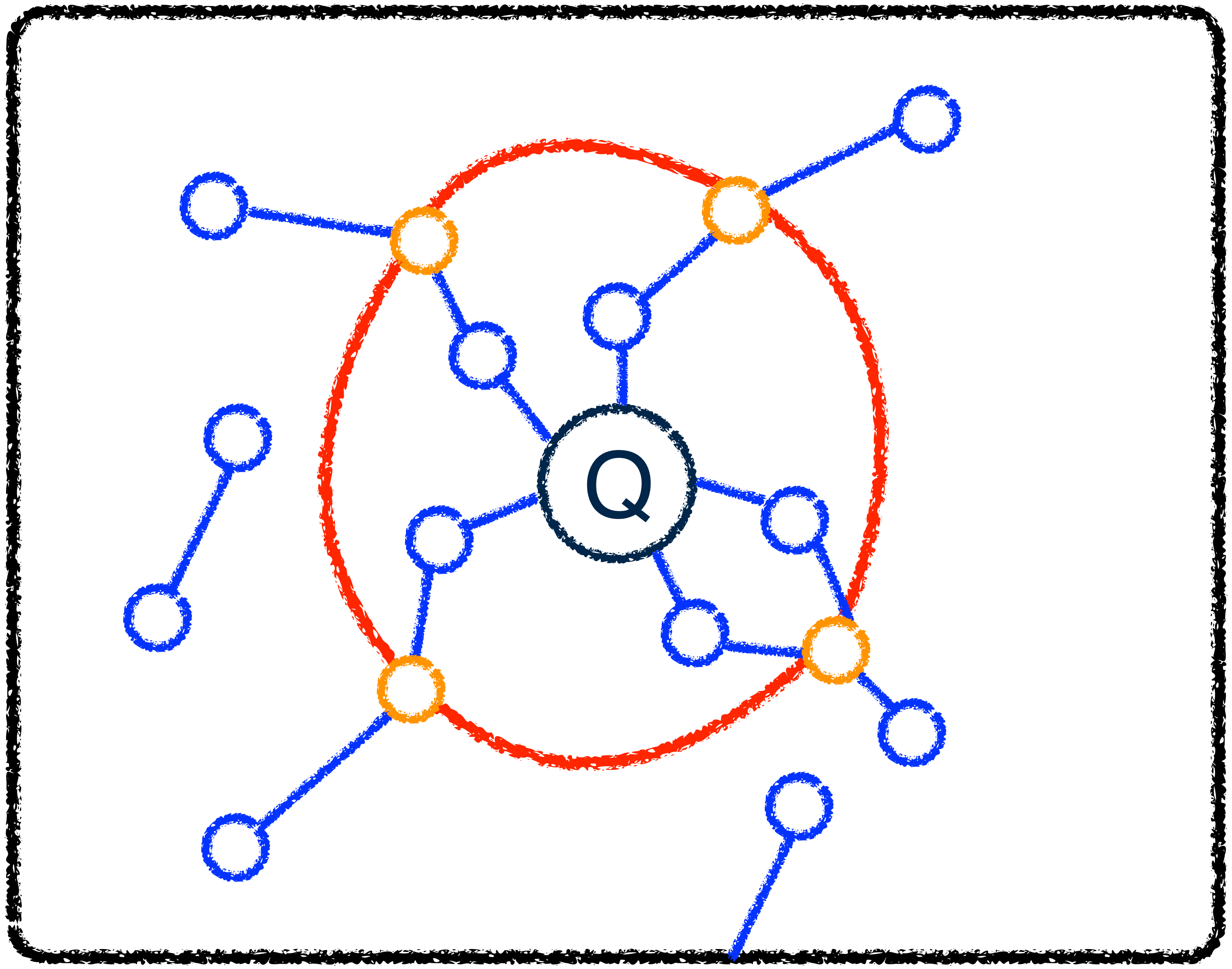
$$\forall \kappa \in \mathsf{K}(\mathsf{fid}(c)).\ \mathsf{MA}_\mathsf{K}(\kappa, c) \in \gamma(\top[x \mapsto \mathsf{PA}_{K_\infty}(c)(x)]).$$

This impact realization holds thanks to two key properties. First, our selective context-sensitivity K (Definition 10) distinguishes all the calling contexts that matter for the queries selected by the pre-analysis. Second, the main analysis designed in Section 4 isolates these distinguished contexts from other undistinguished contexts (ϵ), ensuring that spurious flows caused by merging contexts never adversely affect the precision of the selected query.

Proof Sketch



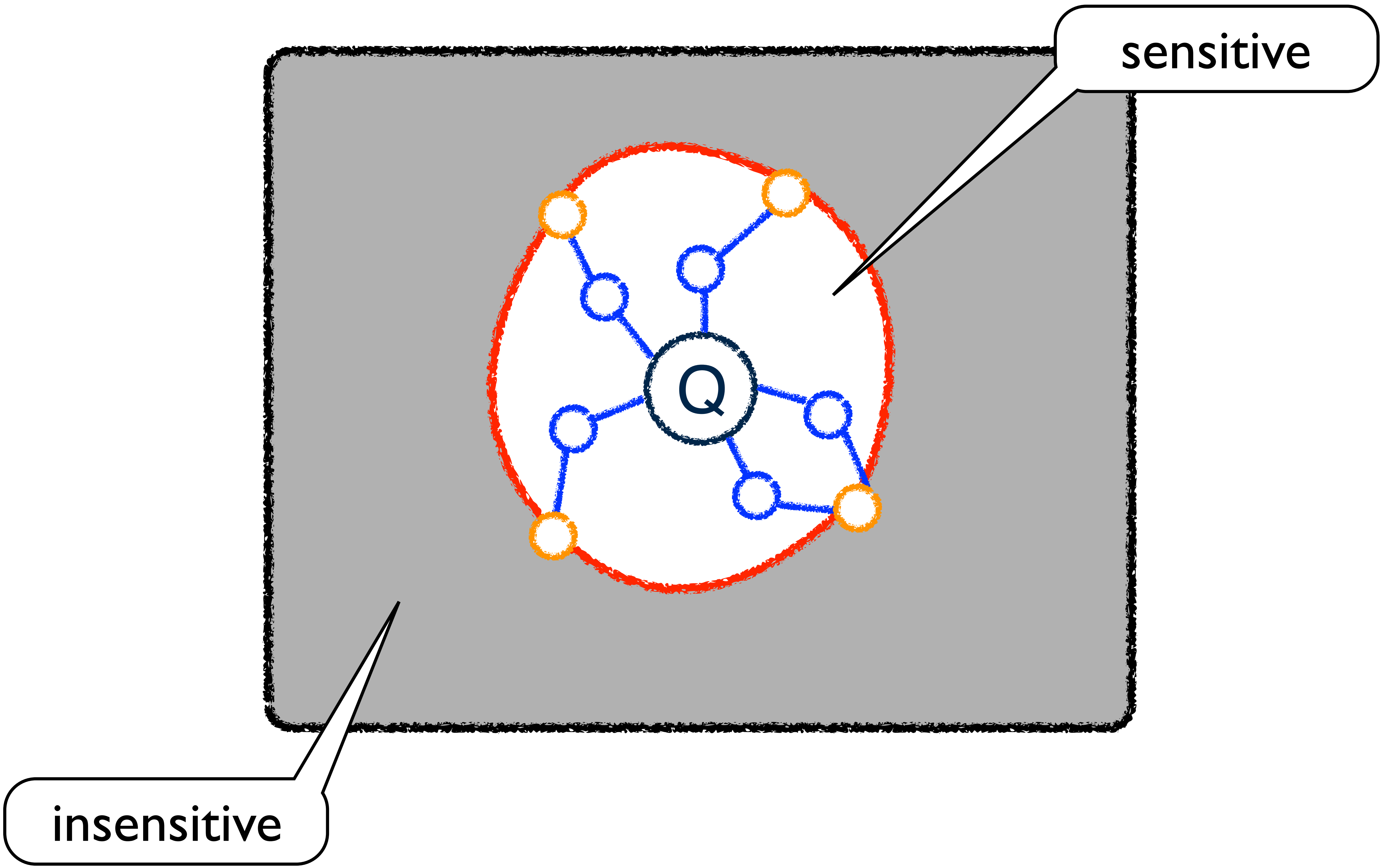
Proof Sketch



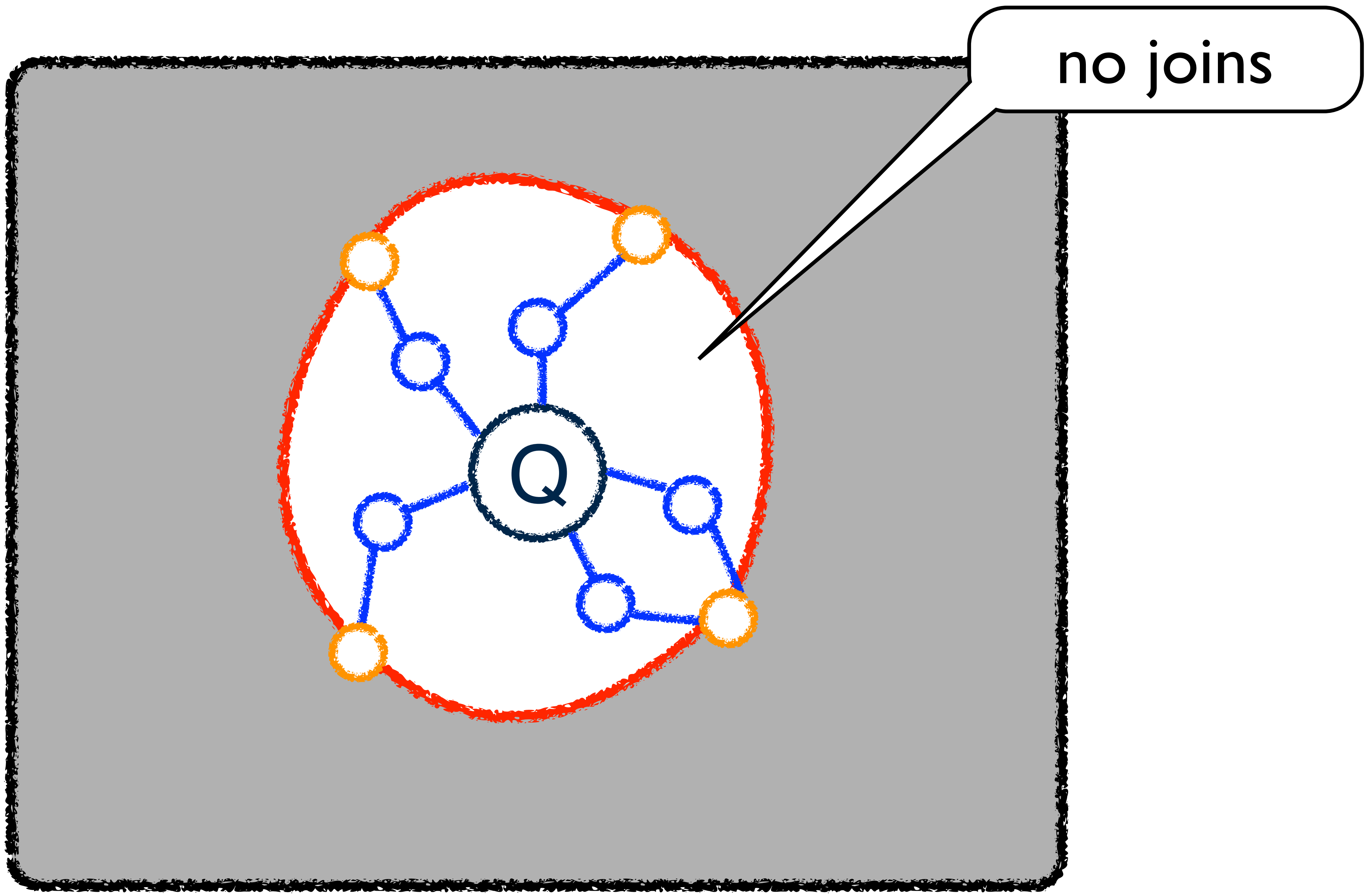
Observation

There exist generators that dominate the query

Proof Sketch

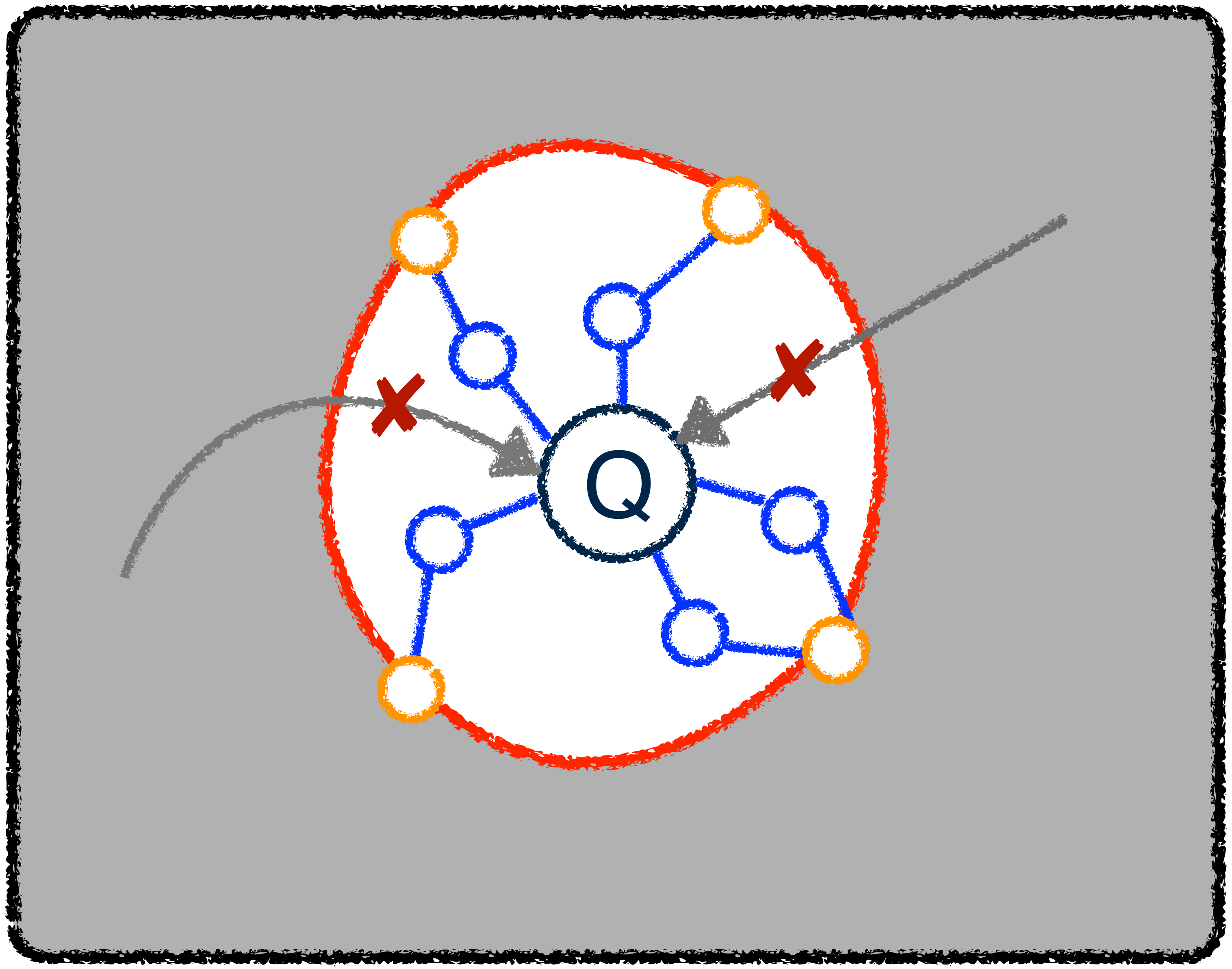


Proof Sketch



1. All generators reach the query without losing precision

Proof Sketch



2. No spurious paths reach the query