Finding Real Bugs in Big Programs with Incorrectness Logic

Paper by: Quang Loc Le et al.
Presented by: Tae Eun Kim

KAIST

Background

Background

Static Analysis

Background

Static Analysis

- Examines the program before runtime

Background

Static Analysis

- Examines the program before runtime
- Scalable compared to dynamic analysis (i.e., Testing)

Background

Static Analysis

- Examines the program before runtime
- Scalable compared to dynamic analysis (i.e., Testing)
  - Considers all possible executions without execution

Background

Static Analysis

- Examines the program before runtime
- Scalable compared to dynamic analysis (i.e., Testing)
    - Considers all possible executions without execution

Infer

Background

Static Analysis

- Examines the program before runtime
- Scalable compared to dynamic analysis (i.e., Testing)
  - Considers all possible executions without execution

Infer

- Static analyzer used in Meta

Background

Static Analysis

- Examines the program before runtime
- Scalable compared to dynamic analysis (i.e., Testing)
  - Considers all possible executions without execution

Infer

- Static analyzer used in Meta
- Compositional analysis

Background

Static Analysis

- Examines the program before runtime
- Scalable compared to dynamic analysis (i.e., Testing)
  - Considers all possible executions without execution

Infer

- Static analyzer used in Meta
- Compositional analysis
  - Analyze parts of the program (functions) first, then combines them

Background

Static Analysis

- Examines the program before runtime
- Scalable compared to dynamic analysis (i.e., Testing)
  - Considers all possible executions without execution

Infer

- Static analyzer used in Meta
- Compositional analysis
  - Analyze parts of the program (functions) first, then combines them
  - Enables diff-time analysis for fast results

Background

Static Analysis

- Examines the program before runtime
- Scalable compared to dynamic analysis (i.e., Testing)
  - Considers all possible executions without execution

Infer

- Static analyzer used in Meta
- Compositional analysis
  - Analyze parts of the program (functions) first, then combines them
  - Enables diff-time analysis for fast results
- Over 100,000 bug reports were fixed

Limitation: Discrepancy

Limitation: Discrepancy

Traditional Static Analysis

Limitation: Discrepancy

Traditional Static Analysis

- "Safe means safe"

Limitation: Discrepancy

Traditional Static Analysis

- “Safe means safe”
- Over-approximates program behavior

Limitation: Discrepancy

Traditional Static Analysis

- “Safe means safe”
- Over-approximates program behavior

```
if (…)
  x = -1; // x = [-1,-1]
else
  x = 1; // x = [1,1]
3/x; // [-1, 1]
```

Limitation: Discrepancy

Traditional Static Analysis

- "Safe means safe"
- Over-approximates program behavior
- Many false alarms, but no misses on true alarms

```
if (…)
  x = -1; // x = [-1,-1]
else
  x = 1; // x = [1,1]
3/x; // [-1, 1]
```

Limitation: Discrepancy

Traditional Static Analysis

- “Safe means safe”
- Over-approximates program behavior
- Many false alarms, but no misses on true alarms

```
if (…)
  x = -1; // x = [-1,-1]
else
  x = 1; // x = [1,1]
3/x; // [-1, 1]
```

Use of Static Analysis in practice

Limitation: Discrepancy

Traditional Static Analysis

- “Safe means safe”
- Over-approximates program behavior
- Many false alarms, but no misses on true alarms

Use of Static Analysis in practice

- “Bug means a bug”

```
1. if (…)
2.   x = -1; // x = [-1,-1]
3. else
4.   x = 1; // x = [1,1]
5. 3/x; // [-1, 1]
```

Limitation: Discrepancy

Traditional Static Analysis

- “Safe means safe”
- Over-approximates program behavior
- Many false alarms, but no misses on true alarms

```
1. if (…)
2.   x = -1; // x = [-1,-1]
3. else
4.   x = 1; // x = [1,1]
5. 3/x; // [-1, 1]
```

Use of Static Analysis in practice

- “Bug means a bug”
- Developers does not want false alarms

Solution

Solution

Pulse-X

Solution

Pulse-X

- Static Analyzer based on Incorrectness Separation Logic (ISL).

Solution

Pulse-X

- Static Analyzer based on Incorrectness Separation Logic (ISL).
- Compositional and Under-approximated analysis

Solution

Pulse-X

- Static Analyzer based on Incorrectness Separation Logic (ISL).
- Compositional and Under-approximated analysis
- 1.5 time higher fix rate over Infer

Solution

Pulse-X

- Static Analyzer based on Incorrectness Separation Logic (ISL).
- Compositional and Under-approximated analysis
- 1.5 time higher fix rate over Infer
- Found 15 new bugs in OpenSSL

Infer

Infer

Hoare Logic

Infer

Hoare Logic

- Base for the Over-approximation

Infer

Hoare Logic

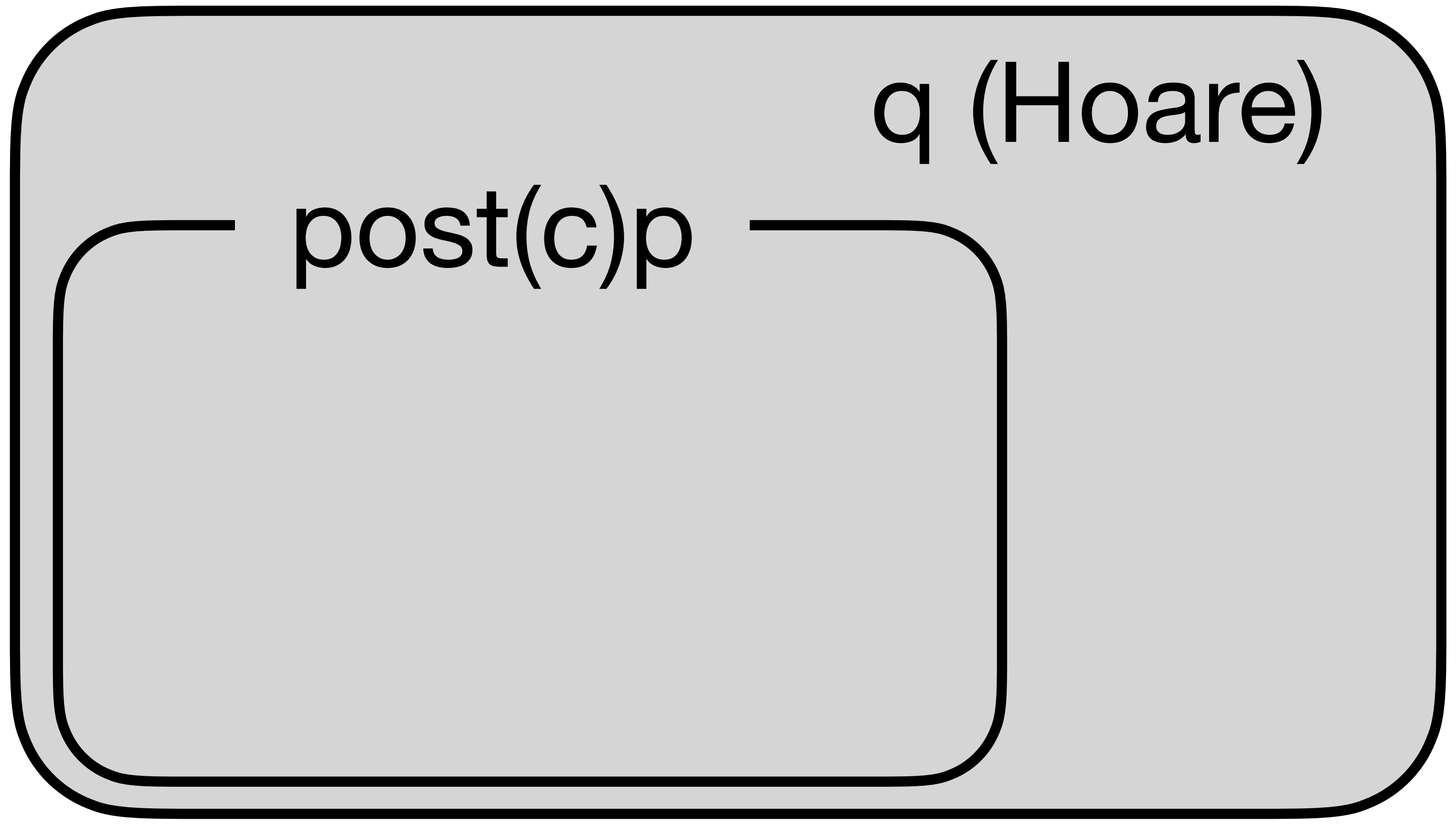
- Base for the Over-approximation
- Hoare triple $\{p\}c\{q\}$ such that, $\{p\}c\{q\} \iff post(c)p \subseteq q$

Infer

Hoare Logic

- Base for the Over-approximation
- Hoare triple $\{p\}c\{q\}$ such that, $\{p\}c\{q\} \iff post(c)p \subseteq q$

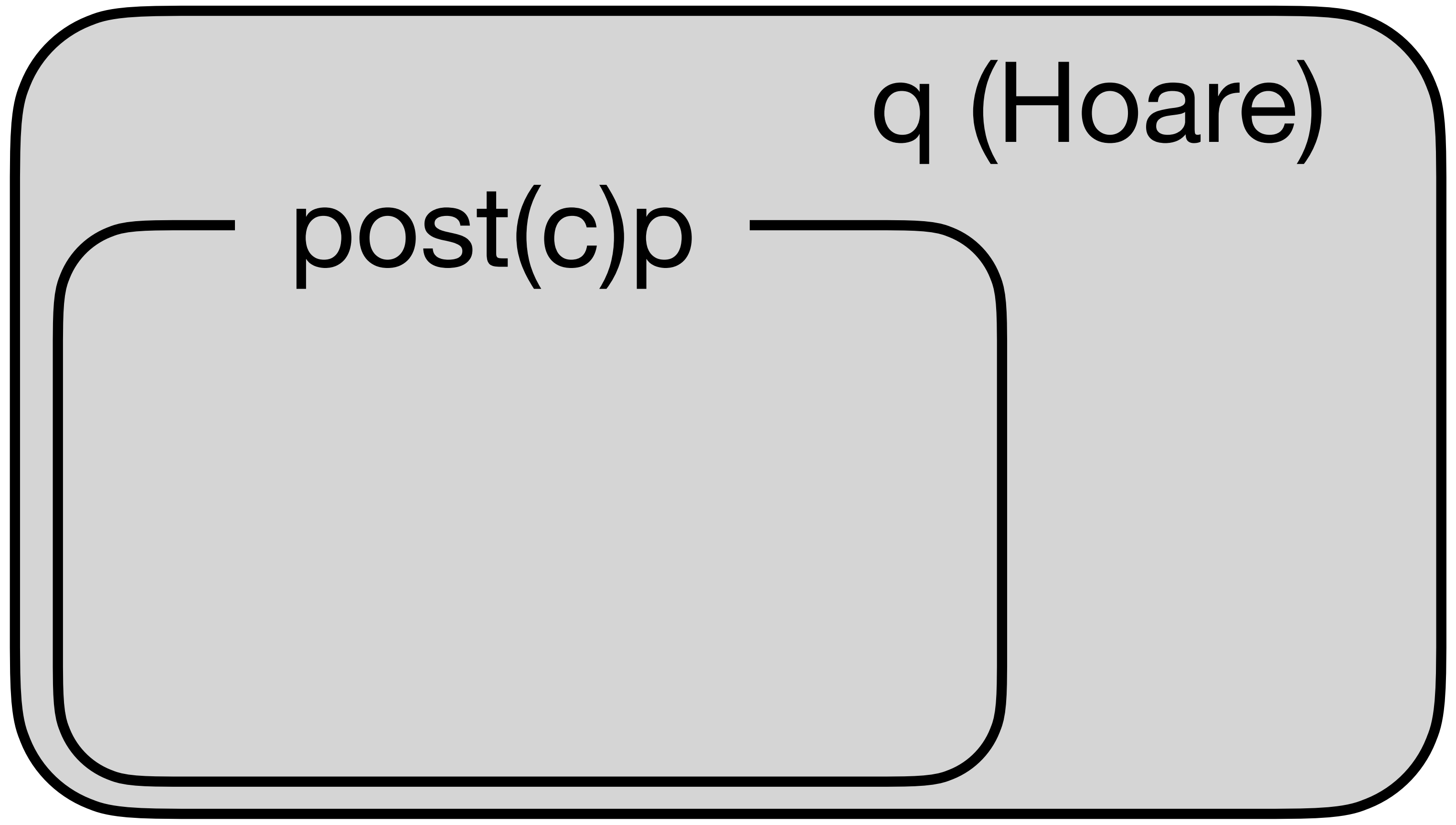
Infer



Hoare Logic

- Base for the Over-approximation
- Hoare triple $\{p\}c\{q\}$ such that, $\{p\}c\{q\} \iff post(c)p \subseteq q$

Infer

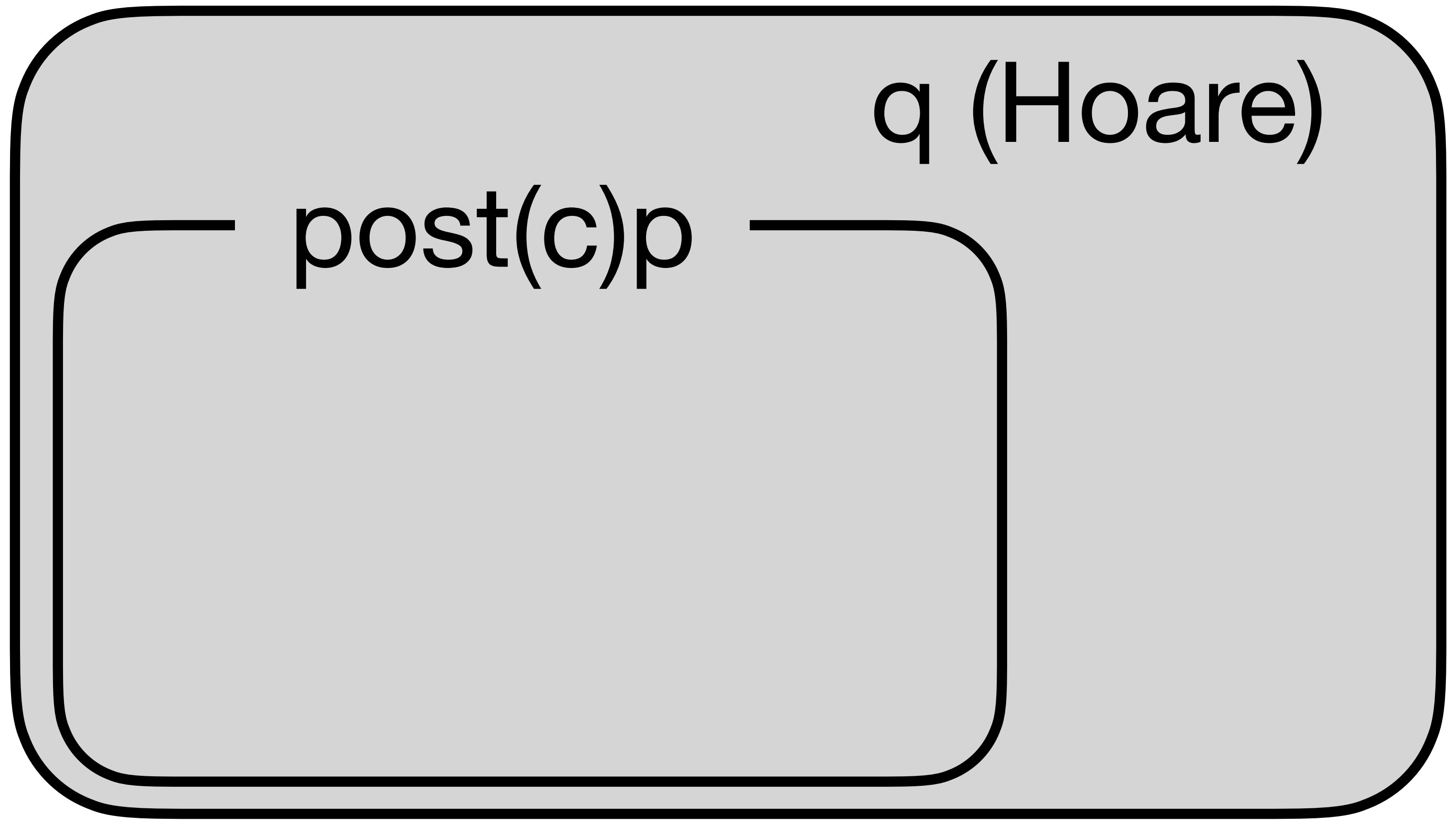


Hoare Logic

- Base for the Over-approximation
- Hoare triple $\{p\}c\{q\}$ such that, $\{p\}c\{q\} \iff post(c)p \subseteq q$

Separational Logic

Infer



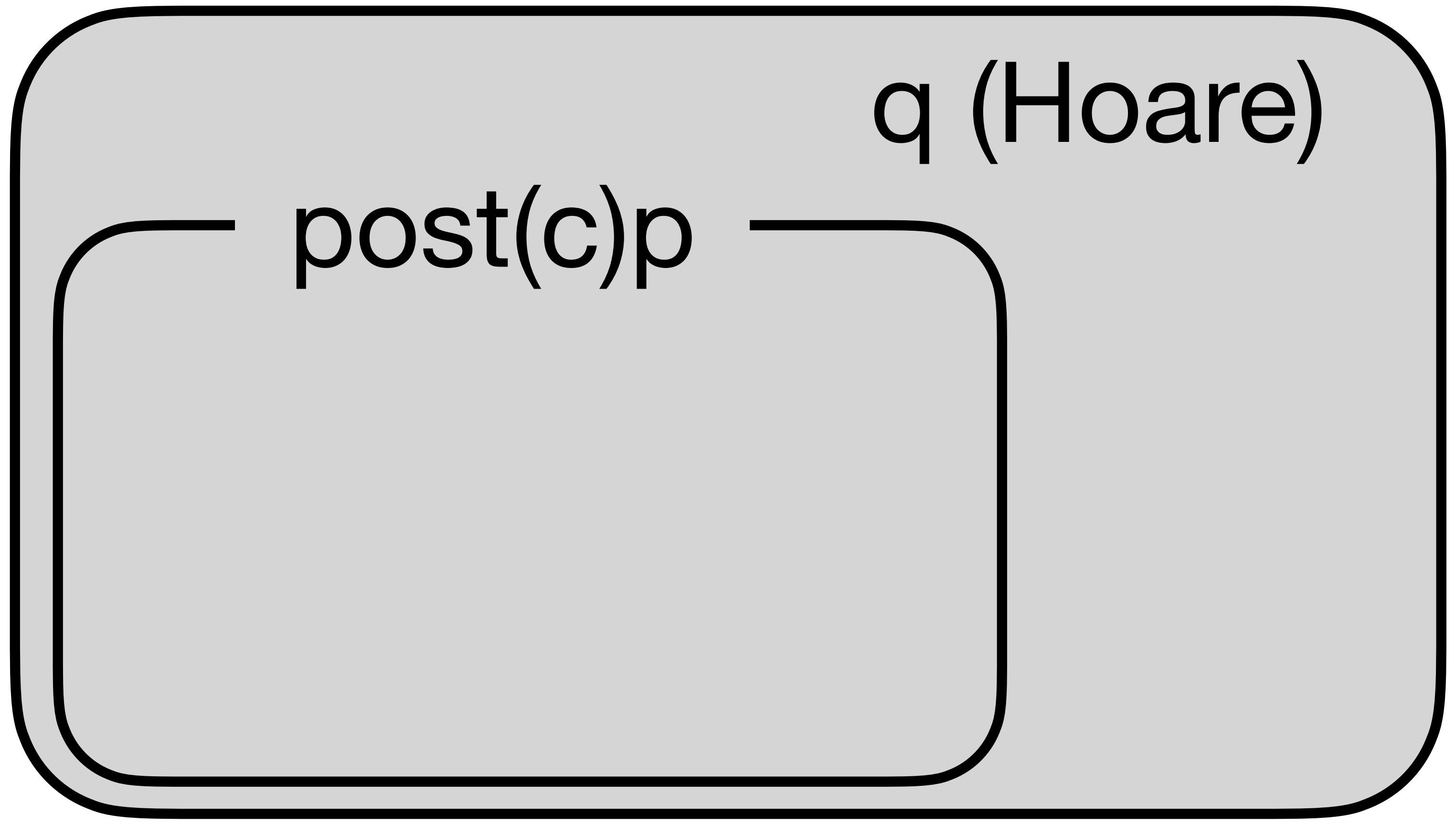
Hoare Logic

- Base for the Over-approximation
- Hoare triple $\{p\}c\{q\}$ such that, $\{p\}c\{q\} \iff post(c)p \subseteq q$

Separational Logic

- Base for the Compositional analysis

Infer



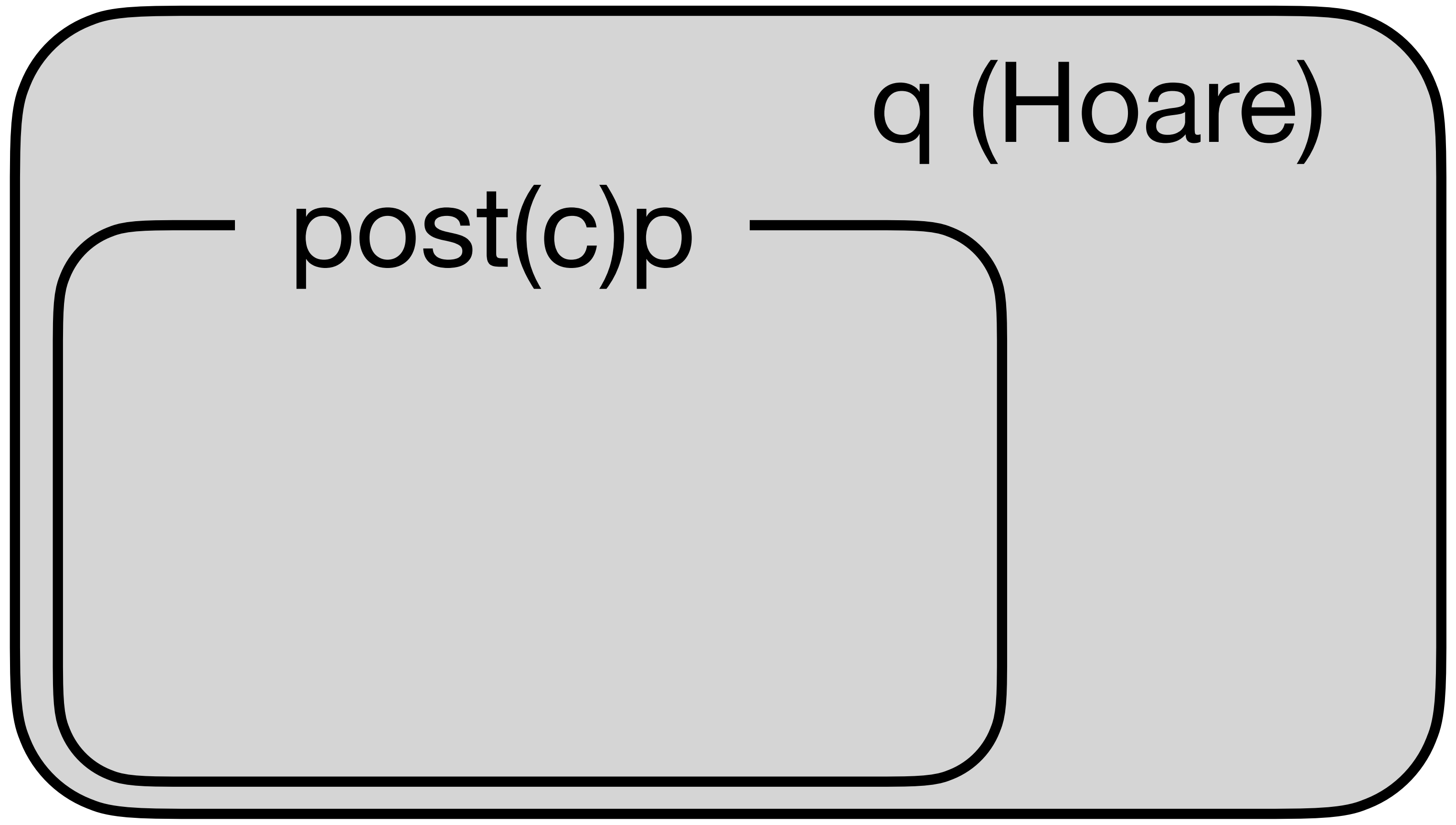
Hoare Logic

- Base for the Over-approximation
- Hoare triple $\{p\}c\{q\}$ such that, $\{p\}c\{q\} \iff post(c)p \subseteq q$

Separational Logic

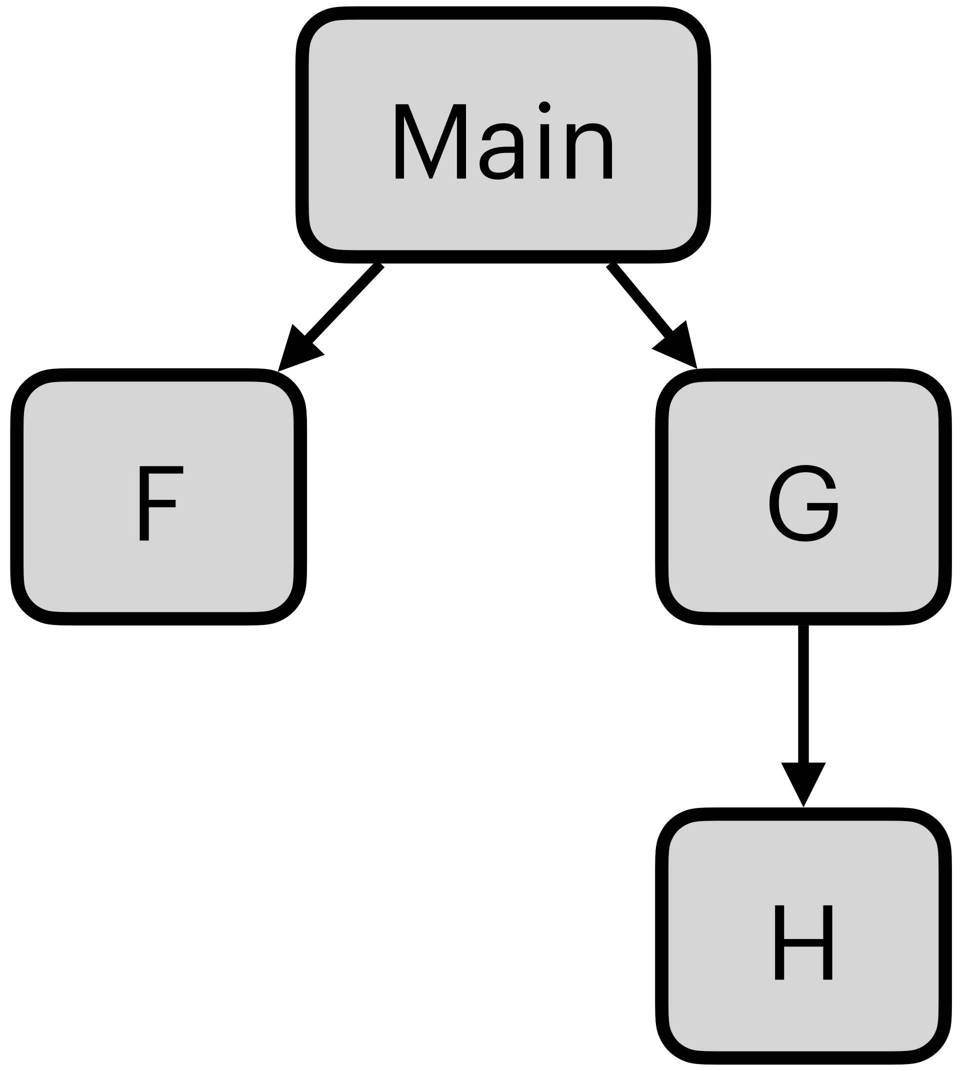
- Base for the Compositional analysis
- Uses Hoare triples as function summaries

Infer



Hoare Logic

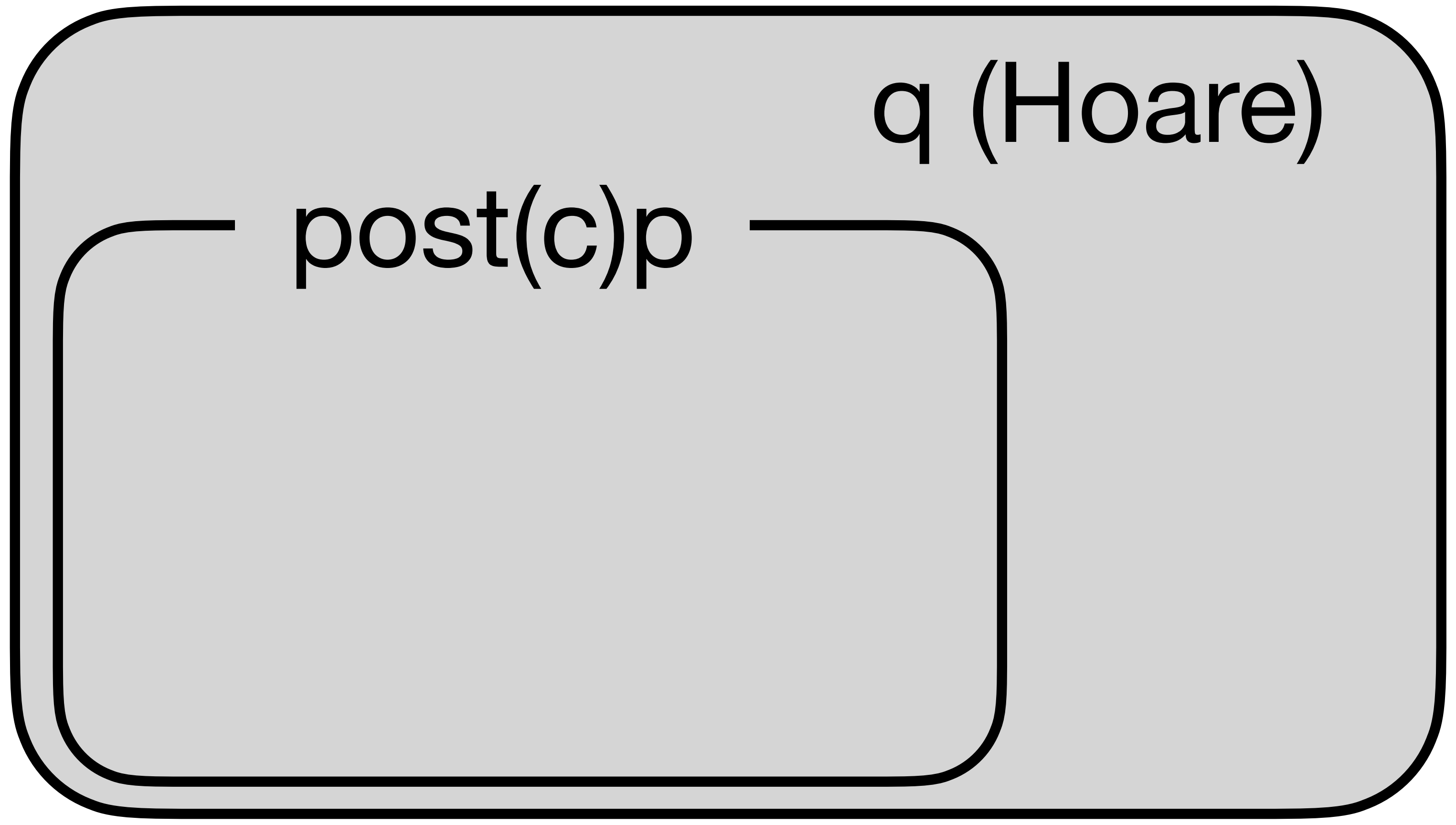
- Base for the Over-approximation
- Hoare triple $\{p\}c\{q\}$ such that, $\{p\}c\{q\} \iff post(c)p \subseteq q$



Separational Logic

- Base for the Compositional analysis
- Uses Hoare triples as function summaries

Infer

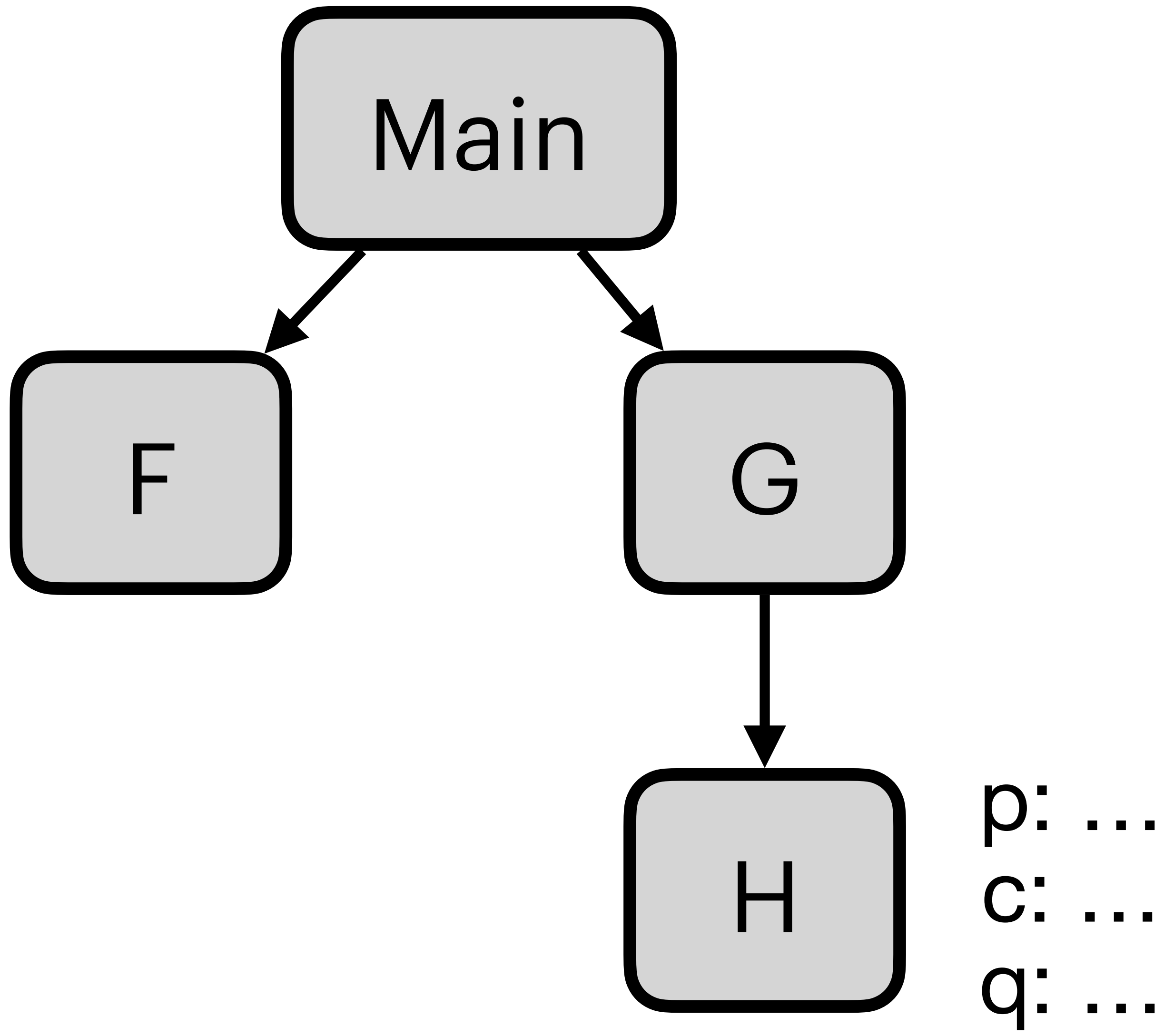


Hoare Logic

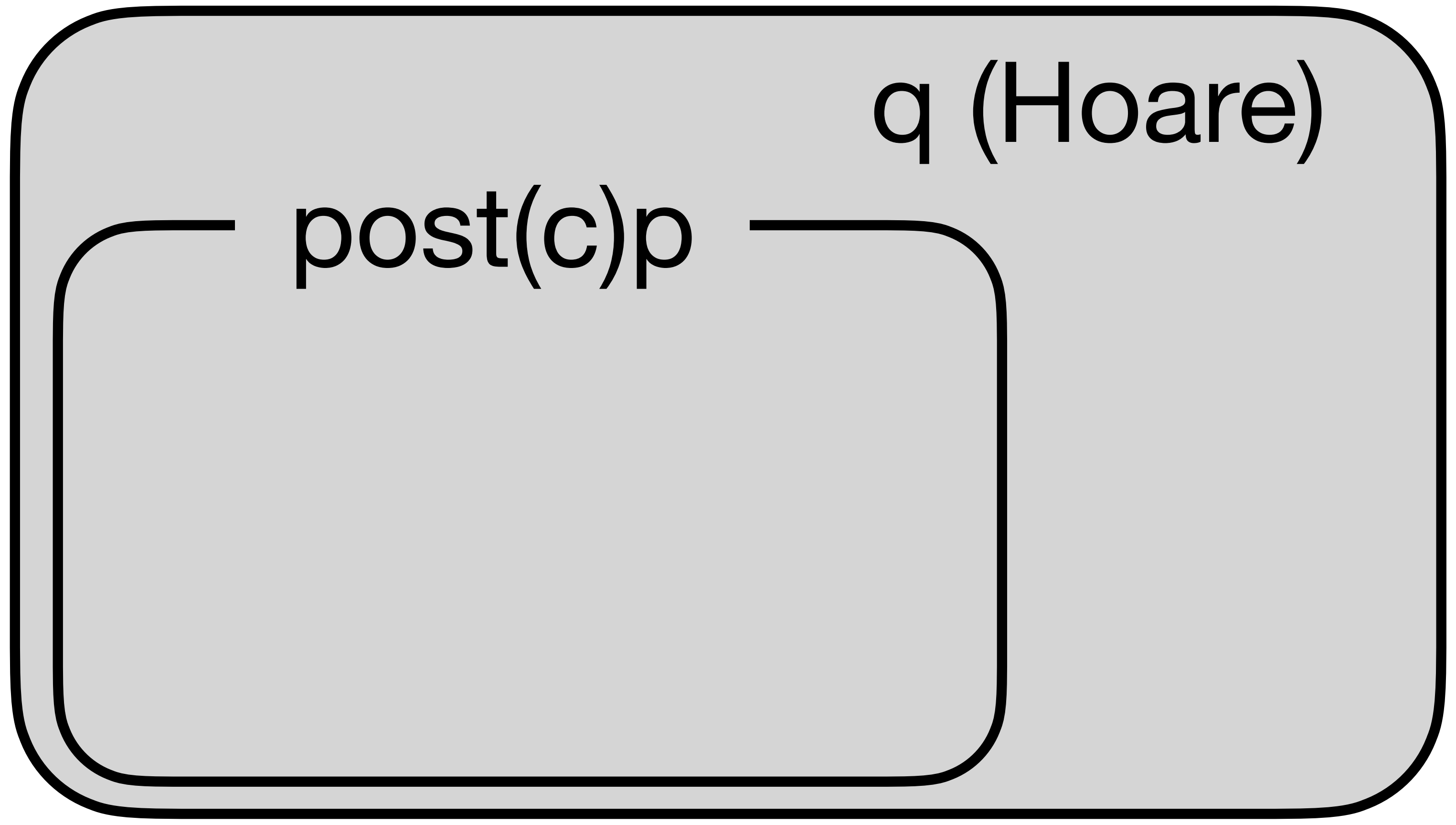
- Base for the Over-approximation
- Hoare triple $\{p\}c\{q\}$ such that, $\{p\}c\{q\} \iff post(c)p \subseteq q$

Separational Logic

- Base for the Compositional analysis
- Uses Hoare triples as function summaries



Infer

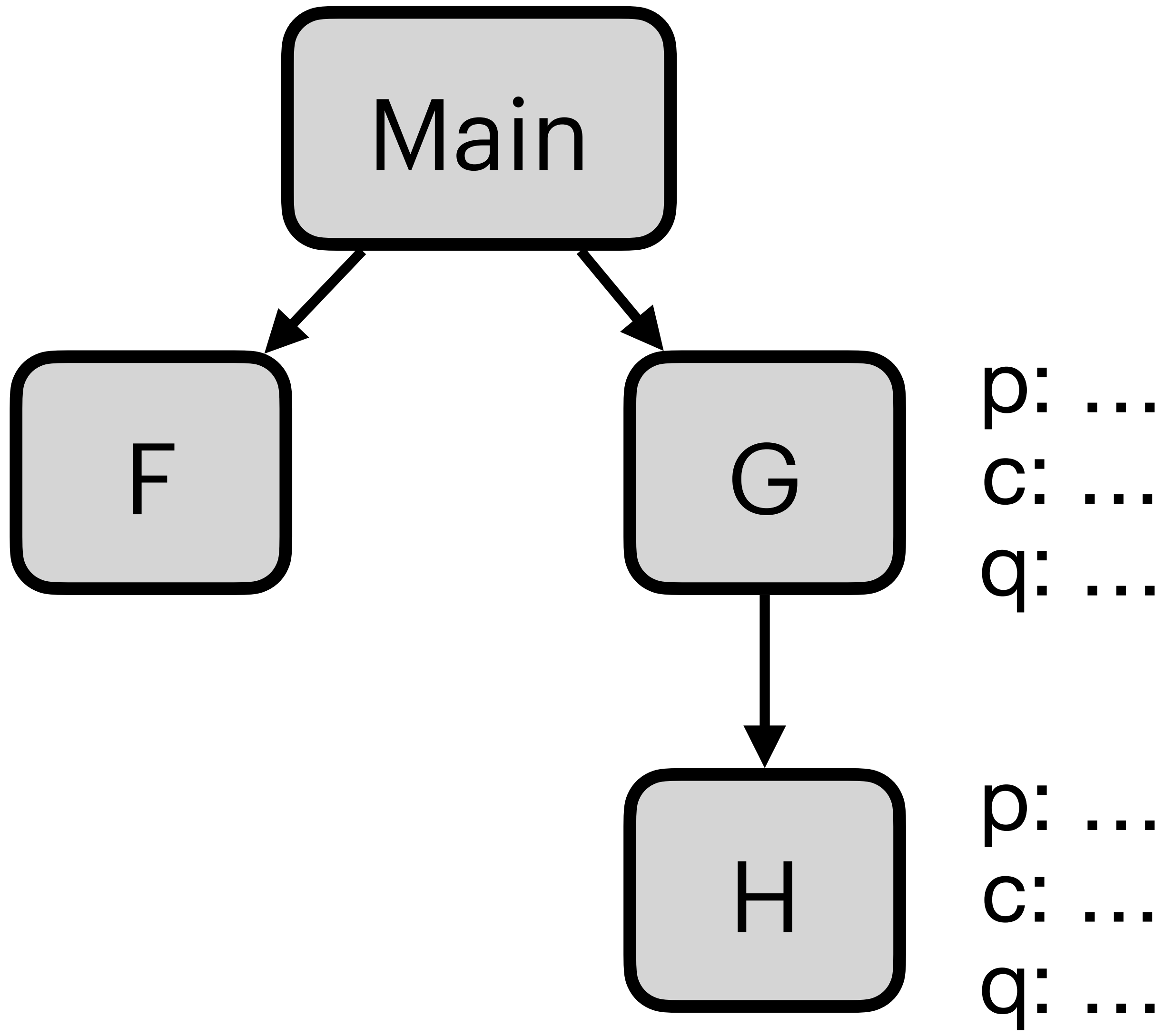


Hoare Logic

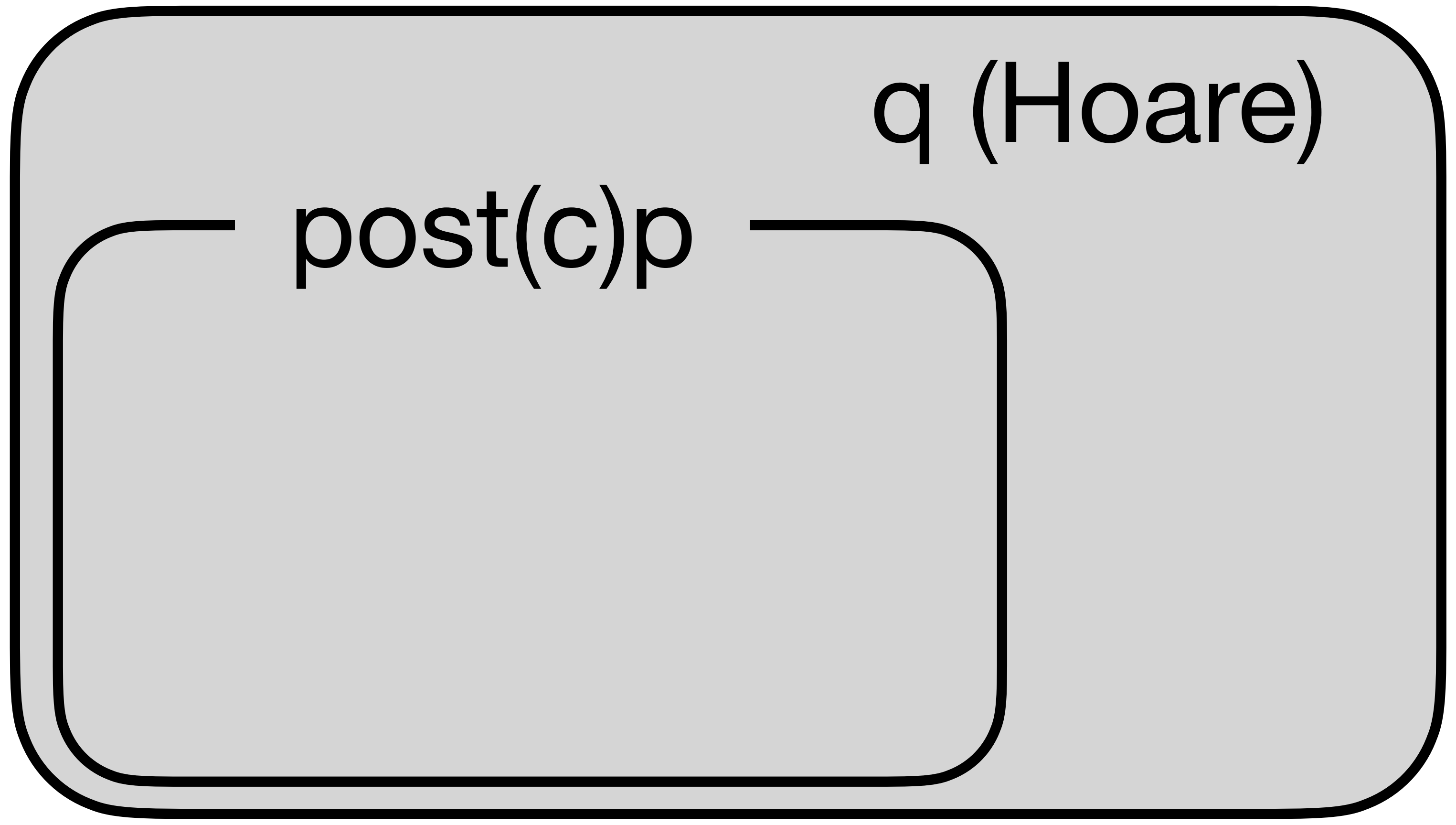
- Base for the Over-approximation
- Hoare triple $\{p\}c\{q\}$ such that, $\{p\}c\{q\} \iff post(c)p \subseteq q$

Separational Logic

- Base for the Compositional analysis
- Uses Hoare triples as function summaries



Infer

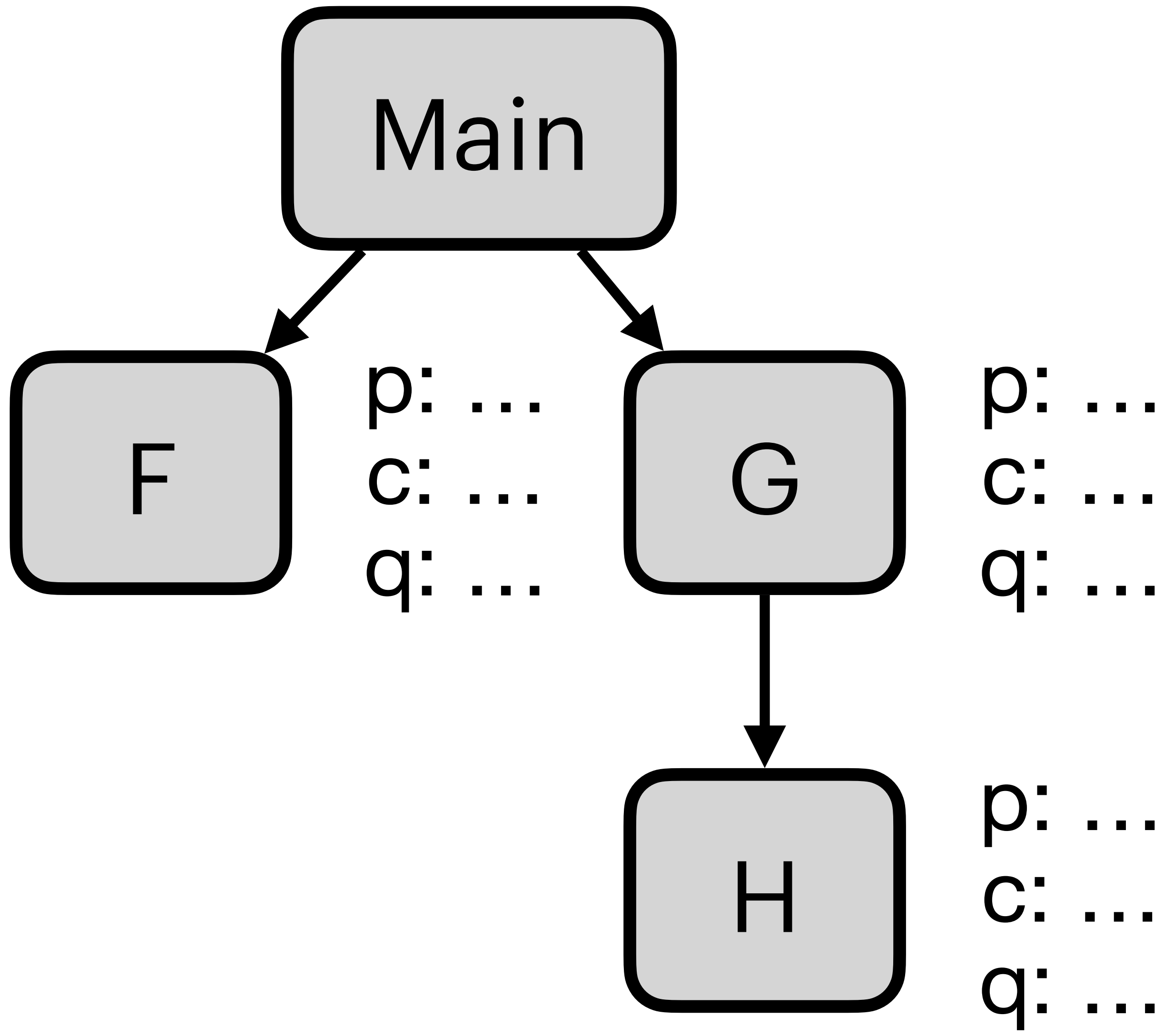


Hoare Logic

- Base for the Over-approximation
- Hoare triple $\{p\}c\{q\}$ such that, $\{p\}c\{q\} \iff post(c)p \subseteq q$

Separational Logic

- Base for the Compositional analysis
- Uses Hoare triples as function summaries



Pulse-X

Pulse-X

Incorrectness Logic (IL)

Pulse-X

Incorrectness Logic (IL)

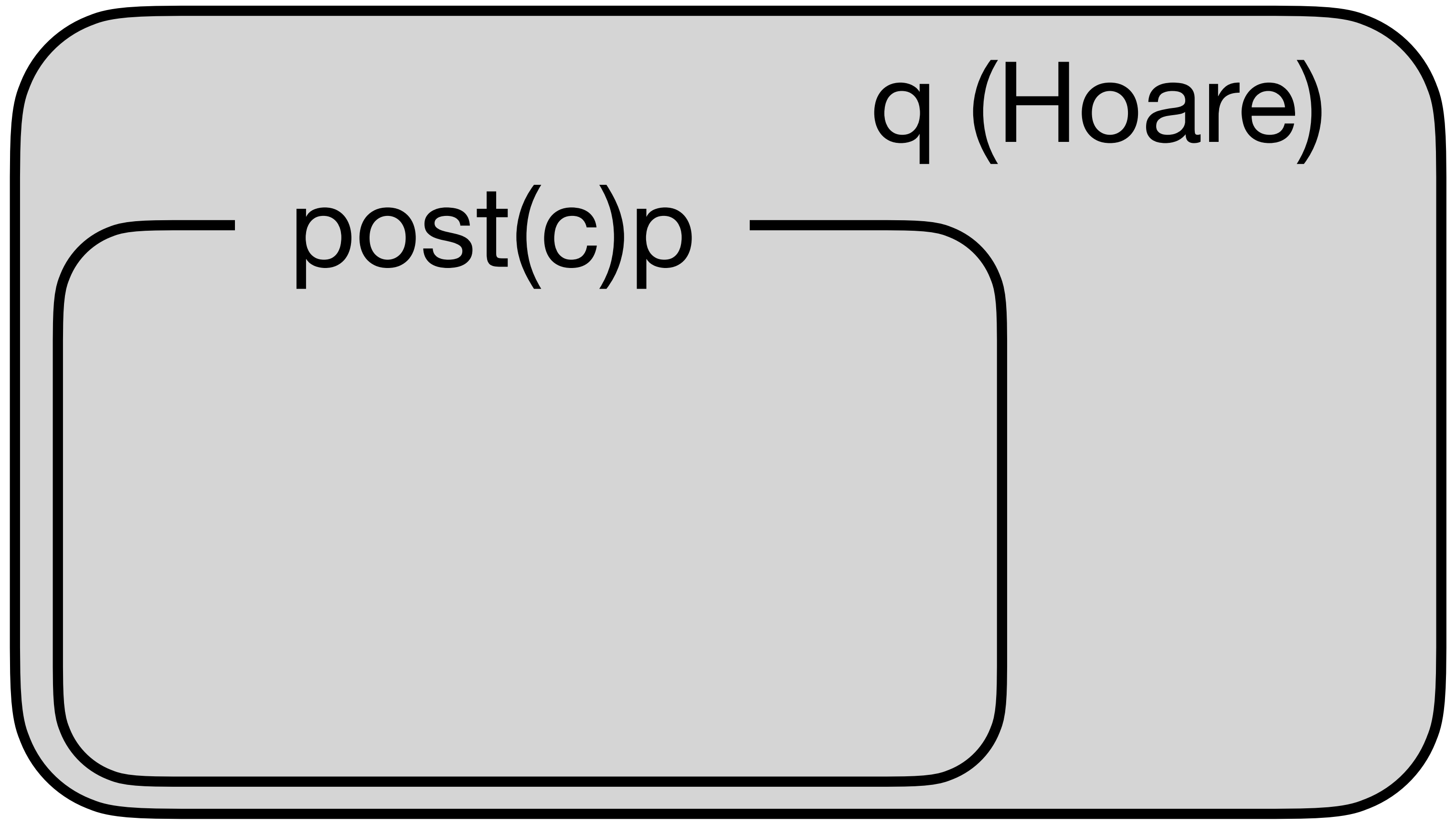
- Base for the Under-approximation

Pulse-X

Incorrectness Logic (IL)

- Base for the Under-approximation
- IL triple $[p]c[q]$ such that, $[p]c[q] \iff post(c)p \supseteq q$

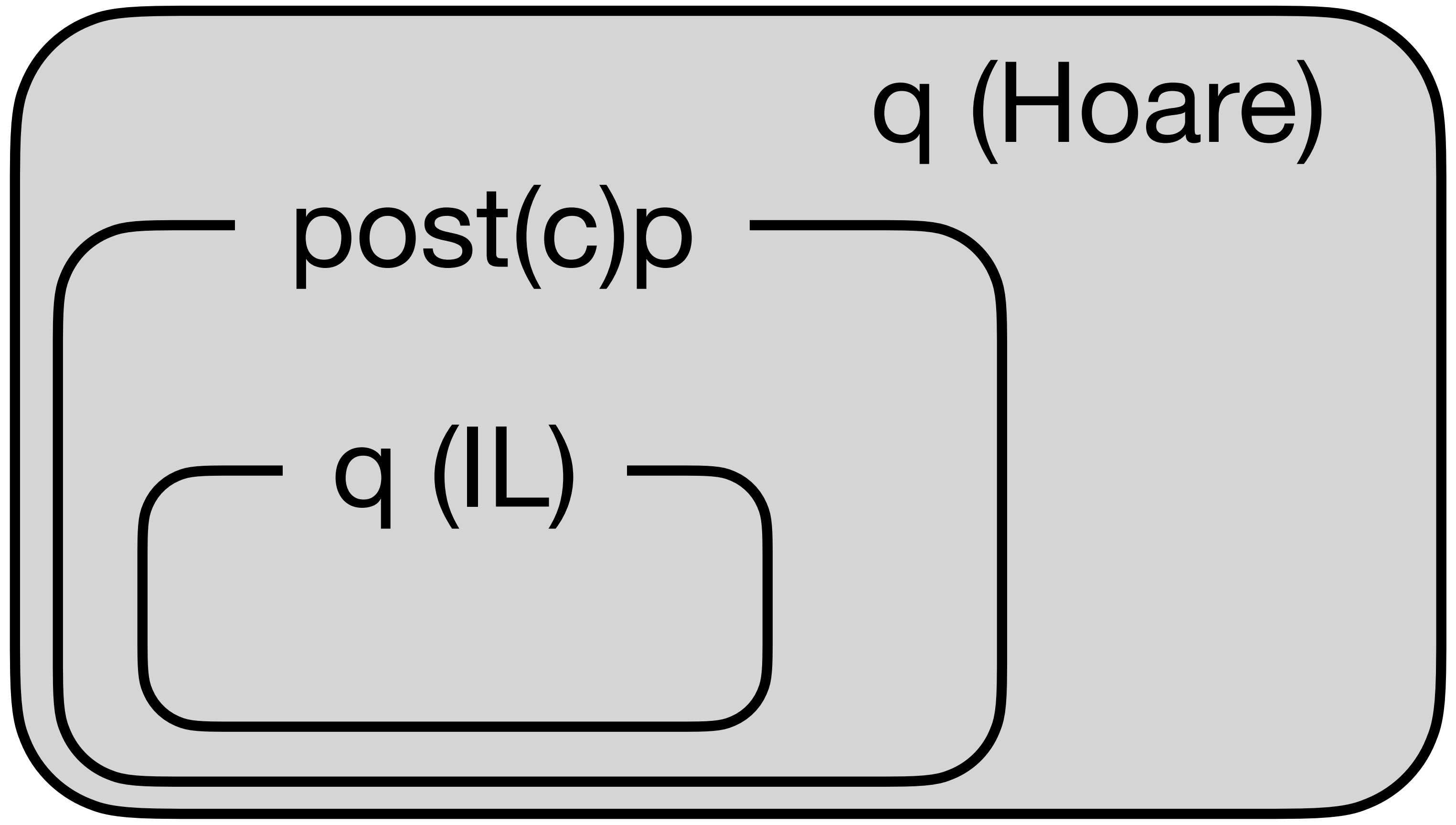
Pulse-X



Incorrectness Logic (IL)

- Base for the Under-approximation
- IL triple $[p]c[q]$ such that, $[p]c[q] \iff post(c)p \supseteq q$

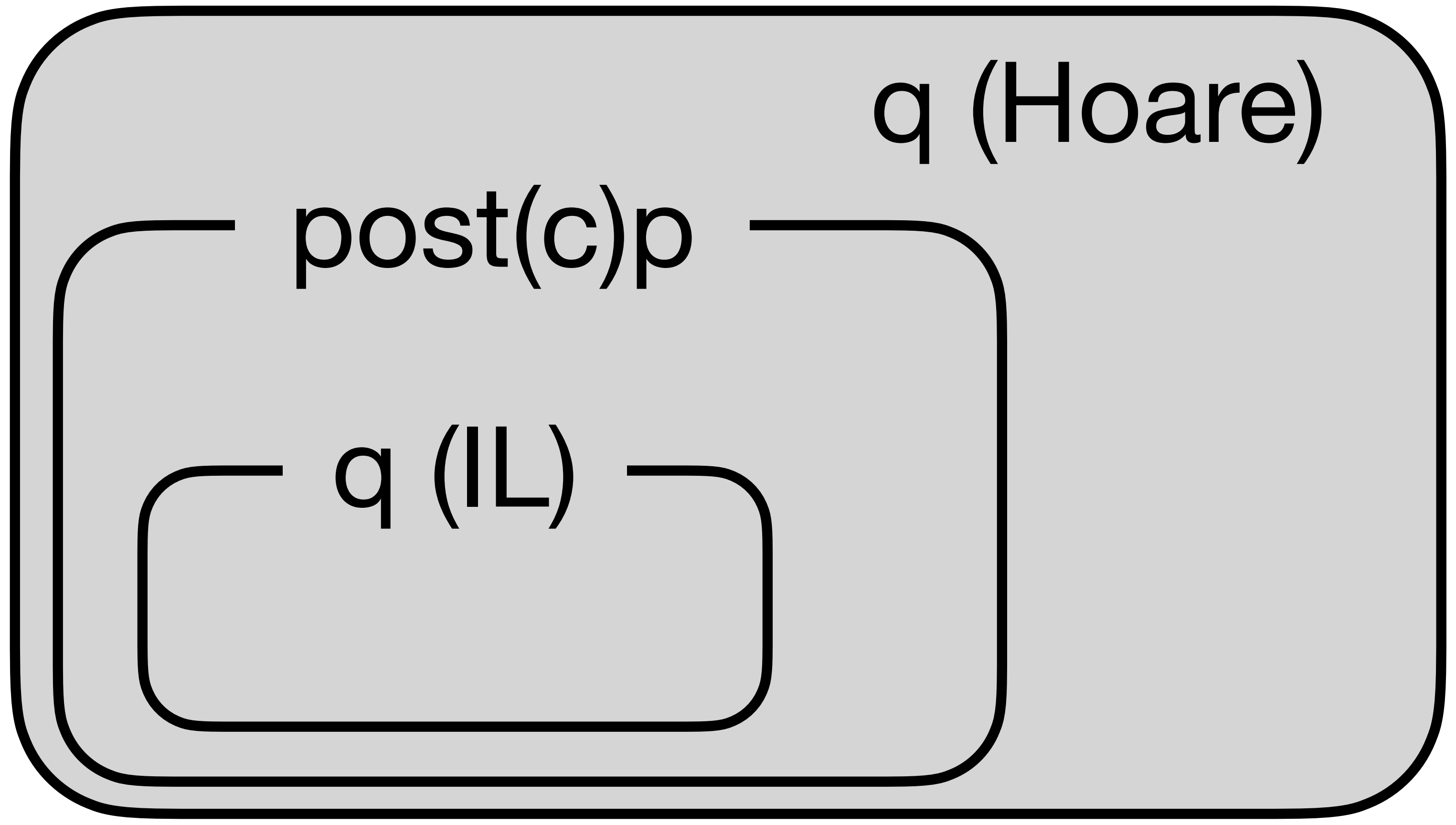
Pulse-X



Incorrectness Logic (IL)

- Base for the Under-approximation
- IL triple $[p]c[q]$ such that, $[p]c[q] \iff post(c)p \supseteq q$

Pulse-X



Incorrectness Logic (IL)

- Base for the Under-approximation
- IL triple $[p]c[q]$ such that, $[p]c[q] \iff post(c)p \supseteq q$
- Add label ($\epsilon \in \{er, ok\}$) to results such that, $[p]c[\epsilon : q]$

$$\left[(\mathop{\ast}_{x_i \in \mathrm{pvars}(\pi)} x_i \mapsto X_i) \land \pi[\overrightarrow{X_i}/\overrightarrow{x_i}]\right] \; \texttt{assume}(\pi) \; \left[ok\colon (\mathop{\ast}_{x_i \in \mathrm{pvars}(\pi)} x_i \mapsto X_i) \land \pi[\overrightarrow{X_i}/\overrightarrow{x_i}]\right]$$

$$\left[\begin{array}{l}(\mathop{\ast}_{y_i \in \mathrm{pvars}(e)\setminus\{x\}} y_i \mapsto Y_i \ast x \mapsto X) \\ \land\, V = e[\overrightarrow{Y_i}/\overrightarrow{y_i}][X/x]\end{array}\right] \; x := e \; \left[ok\colon \begin{array}{l}(\mathop{\ast}_{y_i \in \mathrm{pvars}(e)\setminus\{x\}} y_i \mapsto Y_i \ast x \mapsto V) \\ \land\, V = e[\overrightarrow{Y_i}/\overrightarrow{y_i}][X/x]\end{array}\right]$$

$$[x \mapsto X \ast X \mapsto V \ast y \mapsto Y] \; [x] := y \; [ok\colon x \mapsto X \ast X \mapsto Y \ast y \mapsto Y]$$

$$[x \mapsto X \ast y \mapsto Y \ast Y \mapsto V] \; x := [y] \; [ok\colon x \mapsto V \ast y \mapsto Y \ast Y \mapsto V]$$

$$[\mathsf{emp}] \; \texttt{skip} \; [ok\colon \mathsf{emp}] \qquad [\mathsf{emp}] \; \texttt{error()} \; [er\colon \mathsf{emp}]$$

$$[x \mapsto X \ast X \not\mapsto] \; [x] := y \; [er\colon x \mapsto X \ast X \not\mapsto] \qquad [y \mapsto Y \ast Y \not\mapsto] \; x := [y] \; [er\colon y \mapsto Y \ast Y \not\mapsto]$$

$$[x \mapsto X \land X = \mathsf{nil}] \; [x] := y \; [er\colon x \mapsto X \land X = \mathsf{nil}] \qquad [y \mapsto Y \land Y = \mathsf{nil}] \; x := [y] \; [er\colon y \mapsto Y \land Y = \mathsf{nil}]$$

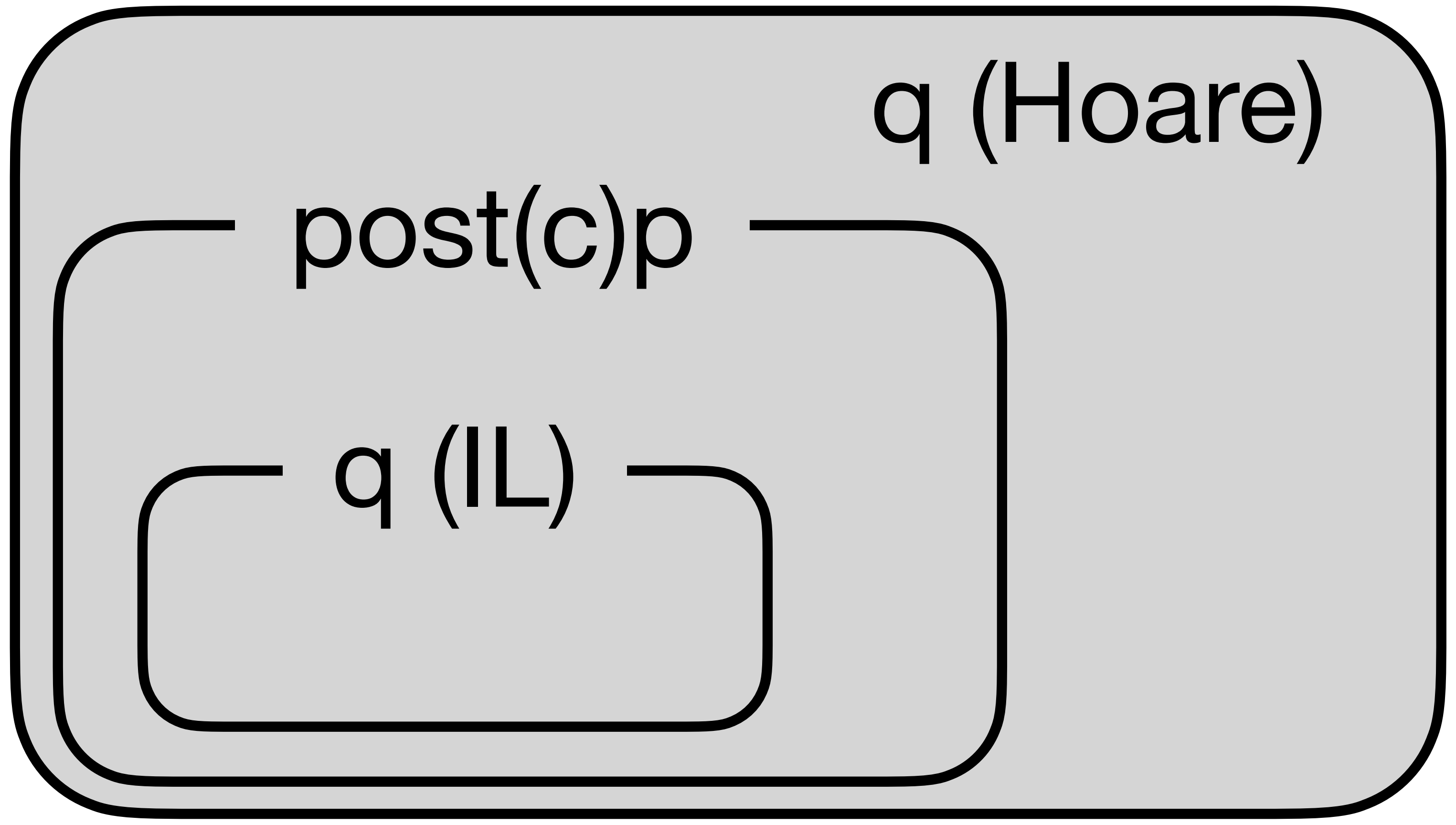
$$[x \mapsto X \ast X \mapsto V] \; \texttt{free}(x) \; [ok\colon x \mapsto X \ast X \not\mapsto] \qquad [x \mapsto X] \; x := \texttt{malloc()} \; [ok\colon \exists L.\, x \mapsto L \ast L \mapsto V \land \mathsf{true}]$$

$$[x \mapsto X \ast X \not\mapsto] \; \texttt{free}(x) \; [er\colon x \mapsto X \ast X \not\mapsto] \qquad [x \mapsto X] \; x := \texttt{malloc()} \; [ok\colon \exists L.\, x \mapsto L \land L = \mathsf{nil}]$$

$$[x \mapsto X \land X = \mathsf{nil}] \; \texttt{free}(x) \; [er\colon x \mapsto X \land X = \mathsf{nil}] \qquad [x \mapsto X \ast \mathsf{ret} \mapsto V] \; \texttt{return}(x) \; [ok\colon x \mapsto X \ast \mathsf{ret} \mapsto X]$$

Fig. 3. Predefined Pulse-X summaries as ISL triples, where pvars(.) returns the program variables of an expression or a statement; for brevity, we omit the pure assertion true and write p in lieu of $p \land \mathsf{true}$.

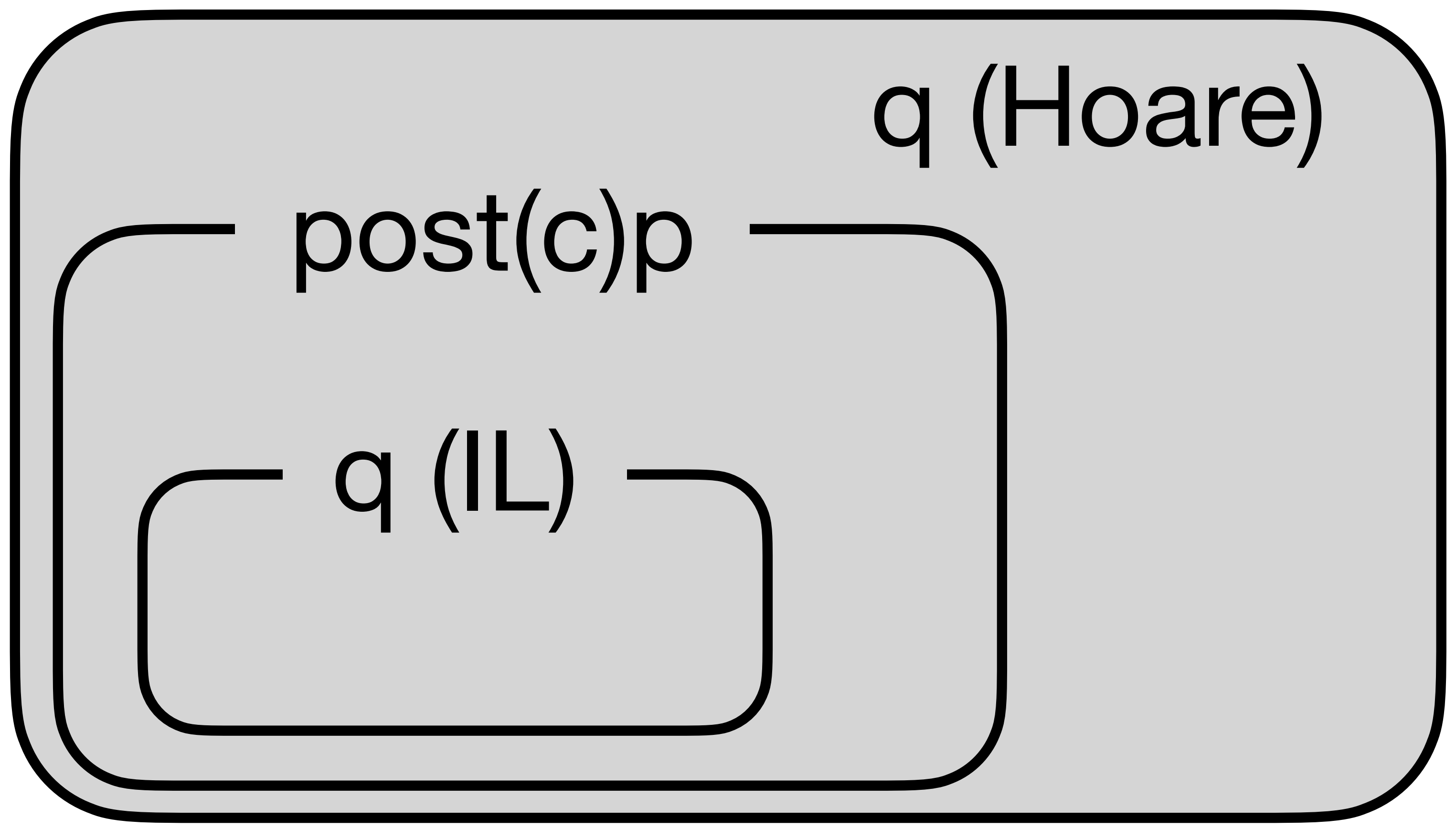
Pulse-X



Incorrectness Logic (IL)

- Base for the Under-approximation
- IL triple $[p]c[q]$ such that, $[p]c[q] \iff post(c)p \supseteq q$
- Add label ($\epsilon \in \{er, ok\}$) to results such that, $[p]c[\epsilon : q]$

Pulse-X

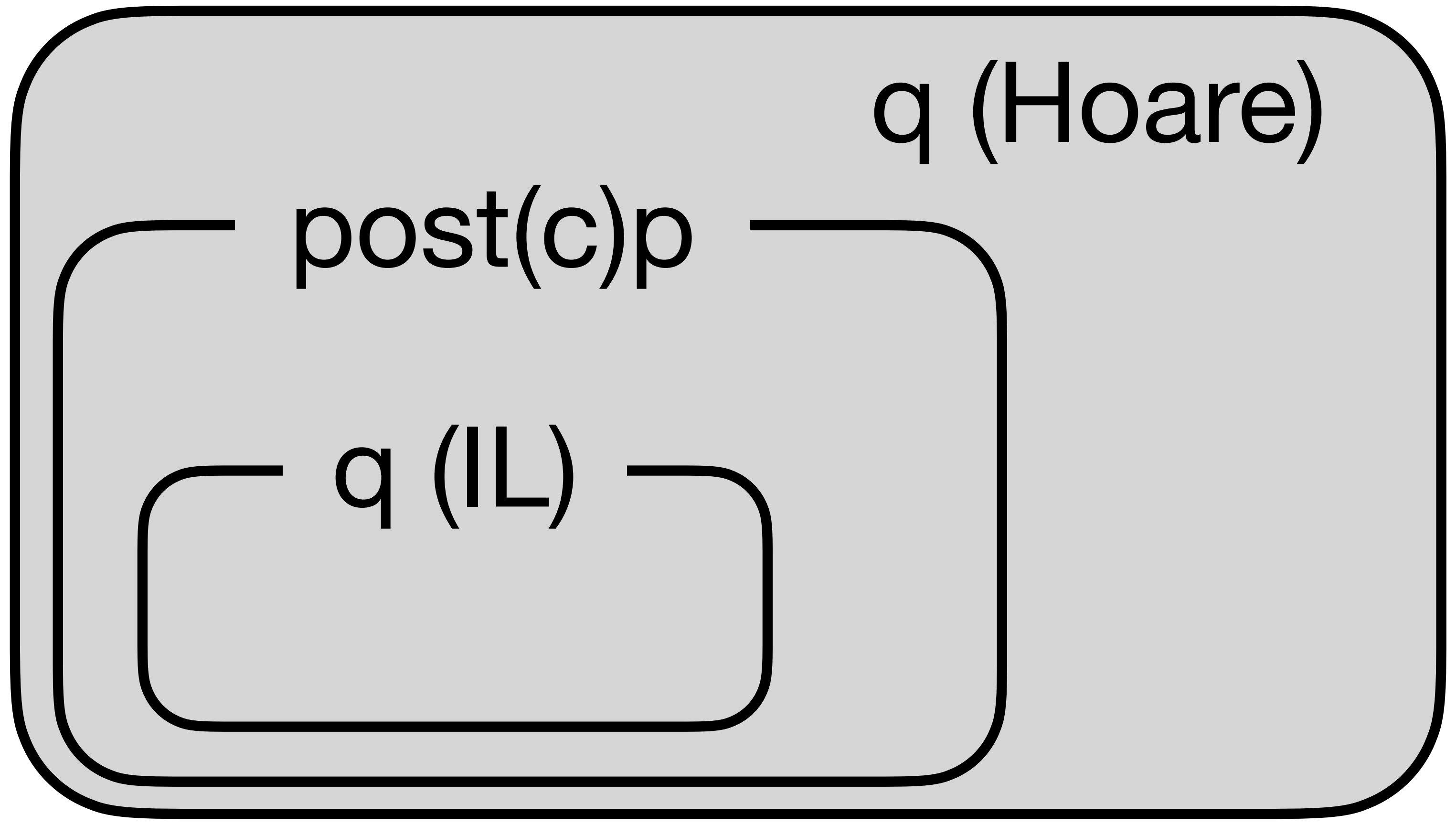


Incorrectness Logic (IL)

- Base for the Under-approximation
- IL triple $[p]c[q]$ such that, $[p]c[q] \iff post(c)p \supseteq q$
- Add label ($\epsilon \in \{er, ok\}$) to results such that, $[p]c[\epsilon : q]$

```
void f (int* x) {
  *x = 42;
}
```

Pulse-X



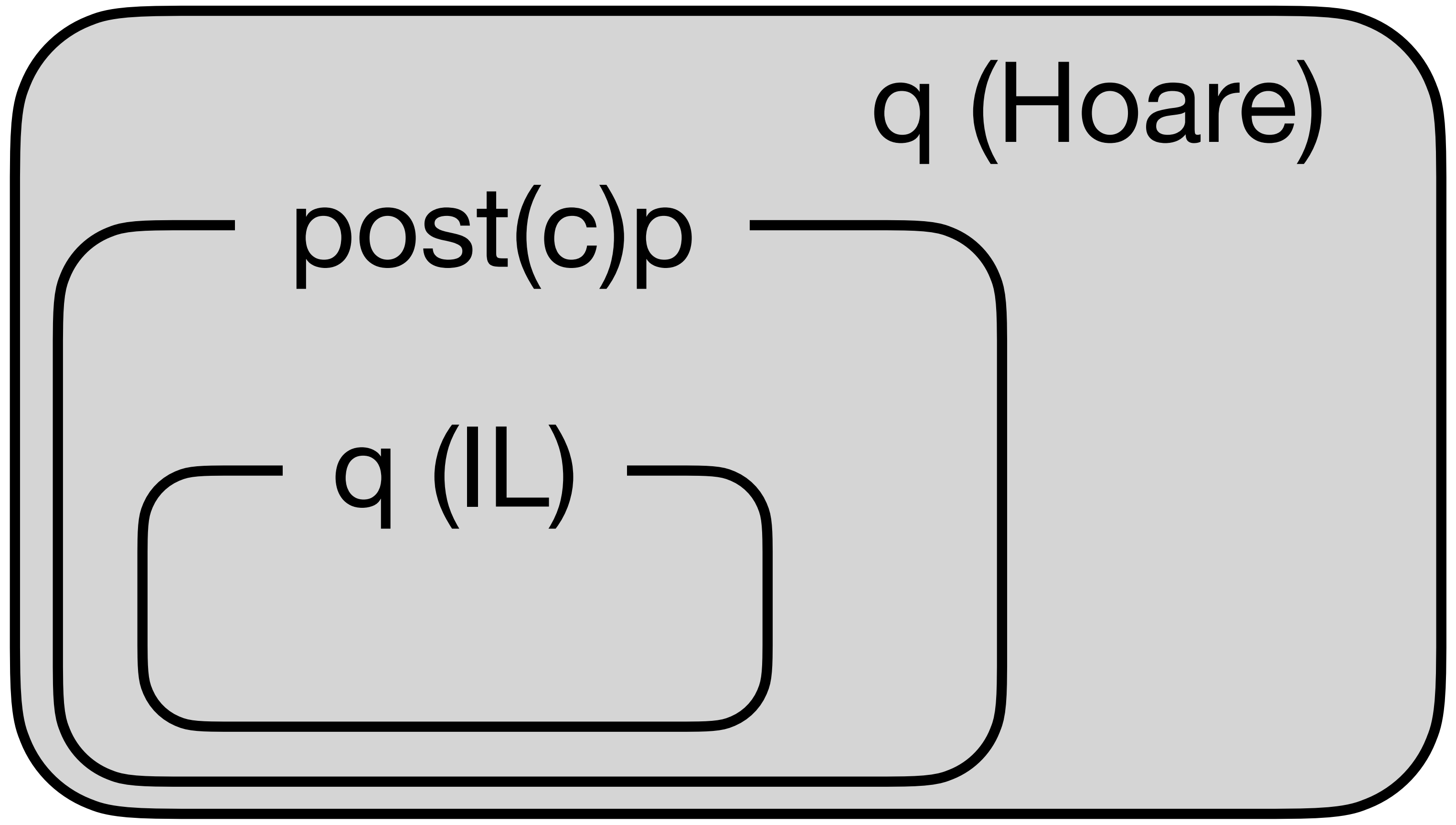
Incorrectness Logic (IL)

- Base for the Under-approximation
- IL triple $[p]c[q]$ such that, $[p]c[q] \iff post(c)p \supseteq q$
- Add label ($\epsilon \in \{er, ok\}$) to results such that, $[p]c[\epsilon : q]$

```
void f (int* x) {
   *x = 42;
}
```

$$[x \mapsto X * X \not\mapsto]\ [x] := y\ [er : x \mapsto X * X \not\mapsto]$$
$$[x \mapsto X \land X = \text{nil}]\ [x] := y\ [er : x \mapsto X \land X = \text{nil}]$$

Pulse-X



Incorrectness Logic (IL)

- Base for the Under-approximation
- IL triple $[p]c[q]$ such that, $[p]c[q] \iff post(c)p \supseteq q$
- Add label ($\epsilon \in \{er, ok\}$) to results such that, $[p]c[\epsilon : q]$

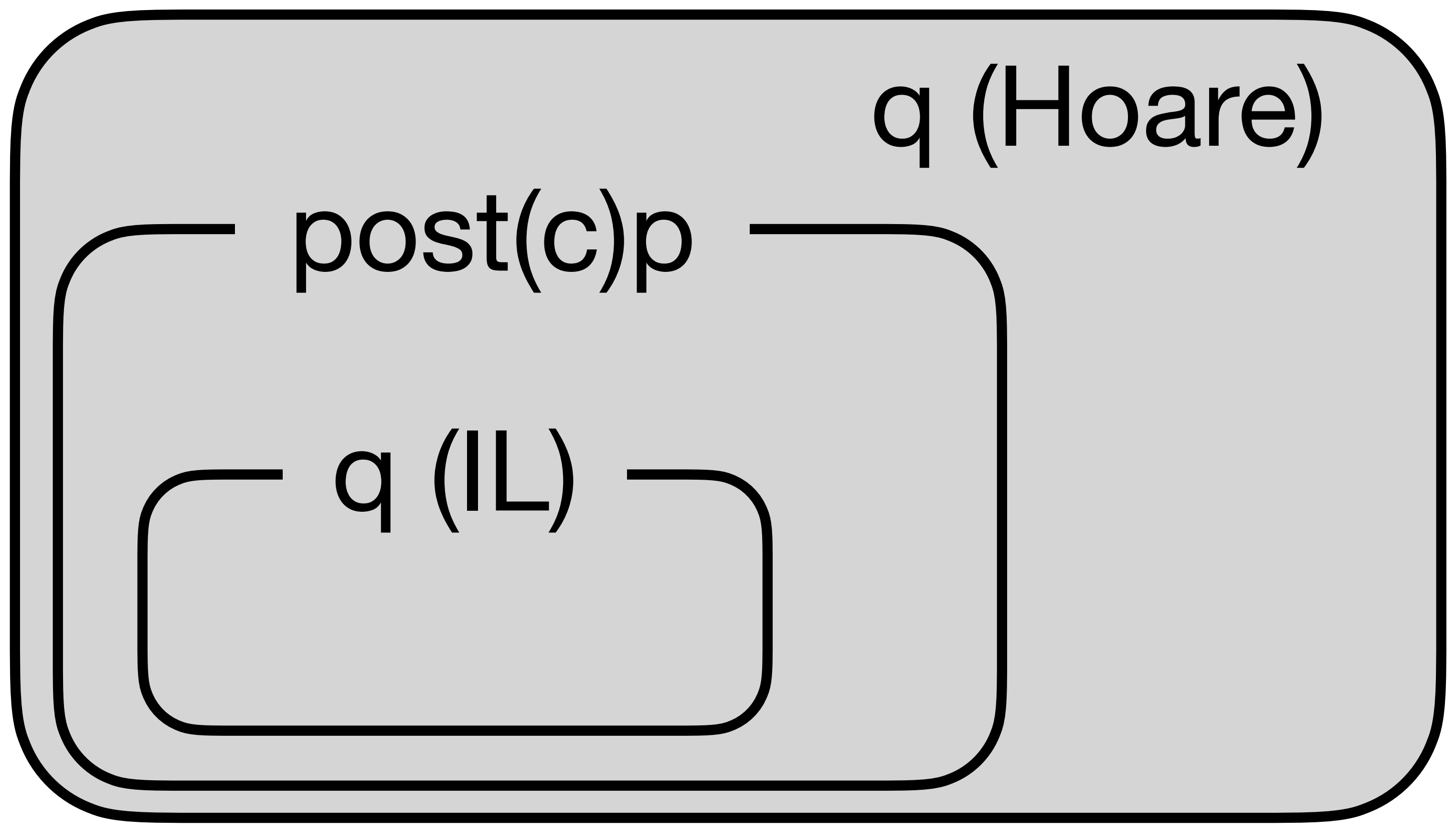
Manifest/Latent Bugs

```
void f (int* x) {
    *x = 42;
}
```

$$[x \mapsto X * X \not\mapsto]\ [x] := y\ [er: x \mapsto X * X \not\mapsto]$$

$$[x \mapsto X \wedge X = \mathsf{nil}]\ [x] := y\ [er: x \mapsto X \wedge X = \mathsf{nil}]$$

Pulse-X



Incorrectness Logic (IL)

- Base for the Under-approximation
- IL triple $[p]c[q]$ such that, $[p]c[q] \iff post(c)p \supseteq q$
- Add label ($\epsilon \in \{er, ok\}$) to results such that, $[p]c[\epsilon : q]$

```
void f (int* x) {
   *x = 42;
}
```

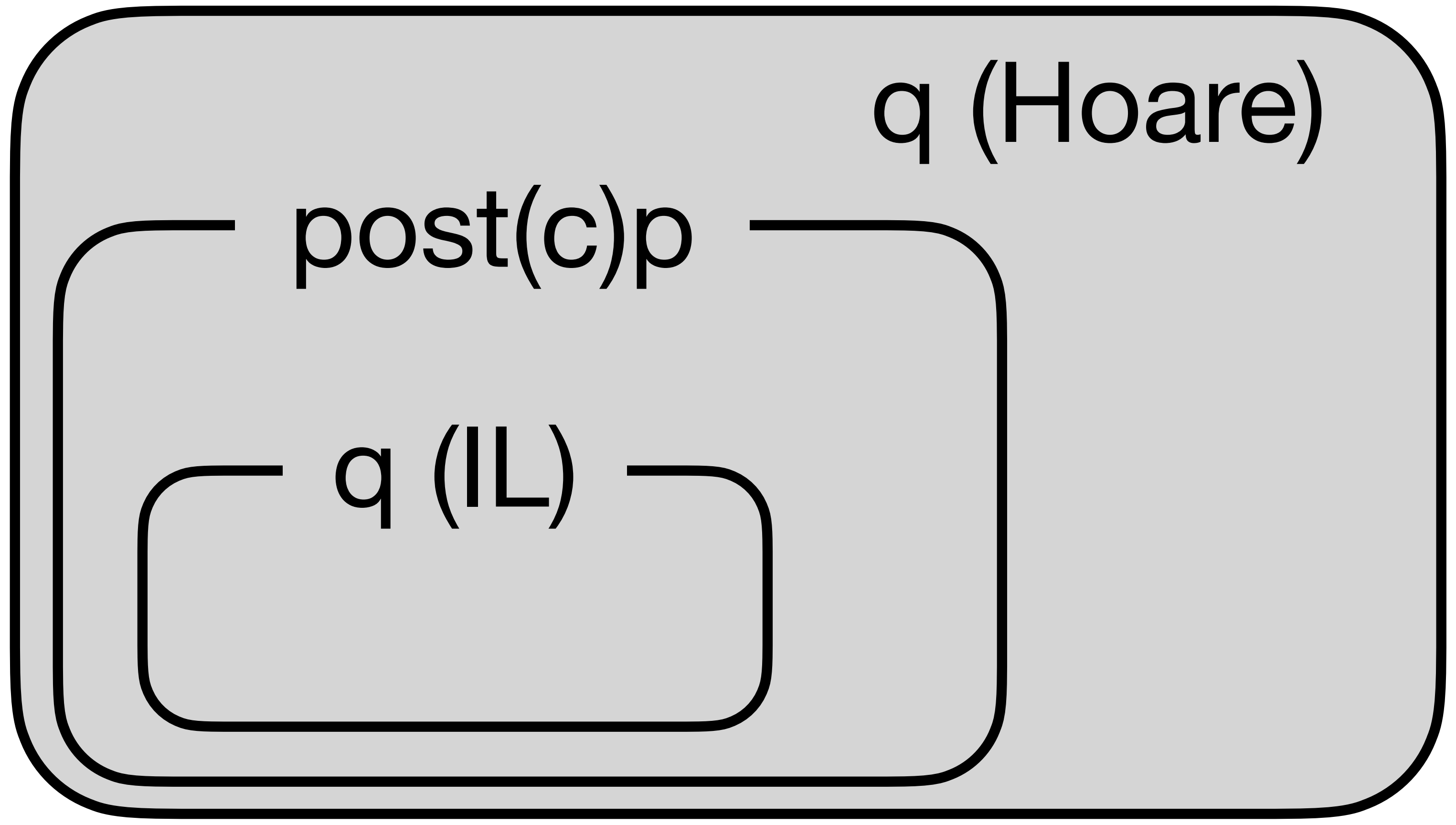
$$[x \mapsto X * X \not\mapsto]\ [x] := y\ [er\colon x \mapsto X * X \not\mapsto]$$

$$[x \mapsto X \land X = \mathrm{nil}]\ [x] := y\ [er\colon x \mapsto X \land X = \mathrm{nil}]$$

Manifest/Latent Bugs

- Another precision filter for the alarms

Pulse-X



Incorrectness Logic (IL)

- Base for the Under-approximation
- IL triple $[p]c[q]$ such that, $[p]c[q] \iff post(c)p \supseteq q$
- Add label ($\epsilon \in \{er, ok\}$) to results such that, $[p]c[\epsilon : q]$

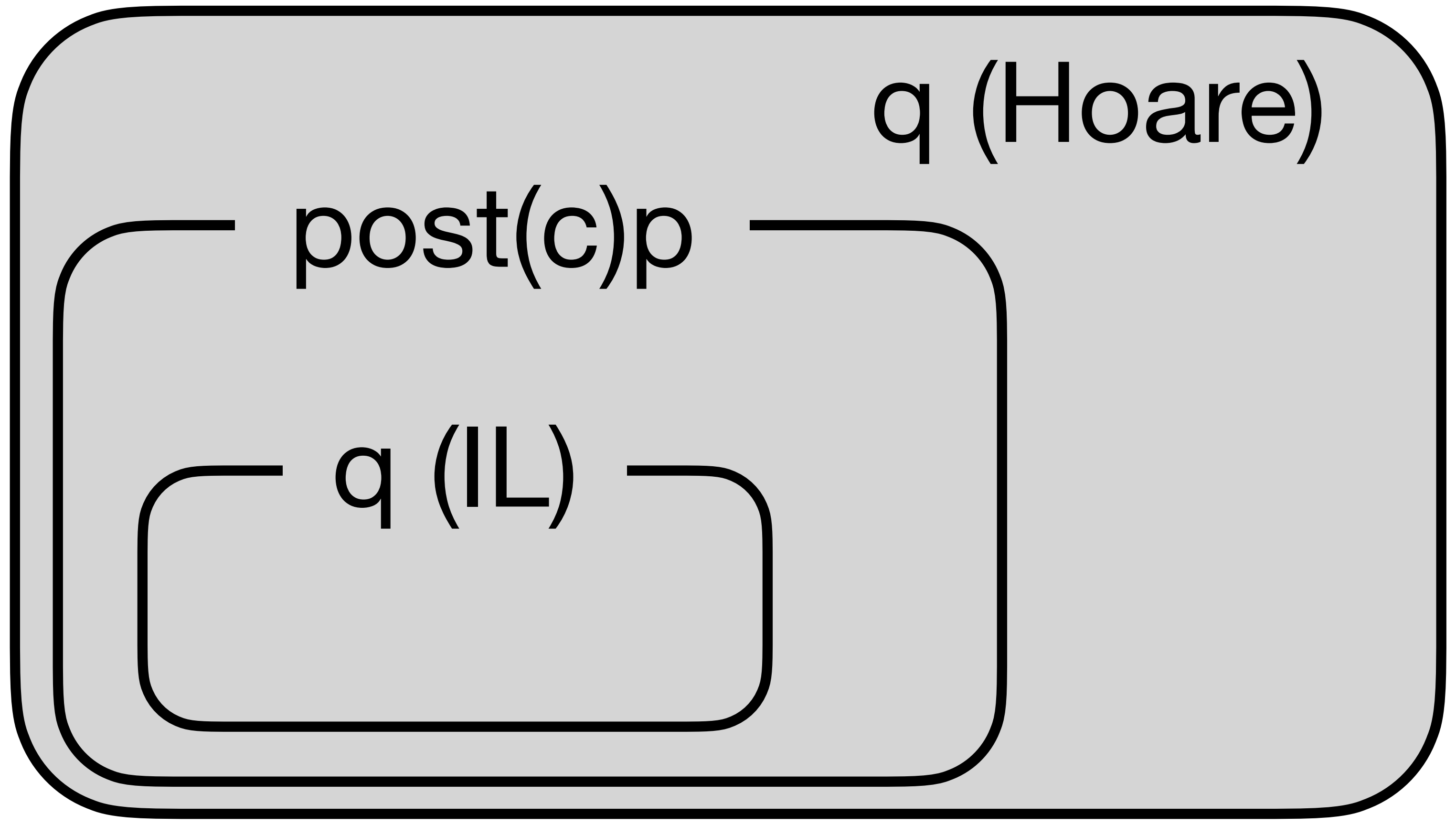
```
void f (int* x) {
  *x = 42;
}
```

$$[x \mapsto X * X \not\mapsto]\ [x] := y\ [er: x \mapsto X * X \not\mapsto]$$
$$[x \mapsto X \land X = \mathrm{nil}]\ [x] := y\ [er: x \mapsto X \land X = \mathrm{nil}]$$

Manifest/Latent Bugs

- Another precision filter for the alarms
- Manifest bugs: any call to the function triggers the bug

Pulse-X



Incorrectness Logic (IL)

- Base for the Under-approximation
- IL triple $[p]c[q]$ such that, $[p]c[q] \iff post(c)p \supseteq q$
- Add label ($\epsilon \in \{er, ok\}$) to results such that, $[p]c[\epsilon : q]$

```
void f (int* x) {
  *x = 42;
}
```

$$[x \mapsto X * X \not\mapsto]\ [x] := y\ [er : x \mapsto X * X \not\mapsto]$$

$$[x \mapsto X \land X = \mathrm{nil}]\ [x] := y\ [er : x \mapsto X \land X = \mathrm{nil}]$$

Manifest/Latent Bugs

- Another precision filter for the alarms
- Manifest bugs: any call to the function triggers the bug
- Latent Bugs: some call to the function triggers the bug

Pulse-X: example 1

Pulse-X: example 1

```
int ssl_excert_prepend(SSL_EXCERT **pexc) {
  SSL_EXCERT *exc = app_malloc(sizeof(exc), “...”);
  memset(exc, 0, sizeof(*exc));
  ...
}
```

Pulse-X: example 1

```
int ssl_excert_prepend(SSL_EXCERT **pexc) {
    SSL_EXCERT *exc = app_malloc(sizeof(exc), "...");
    memset(exc, 0, sizeof(*exc));
    ...
}
```

Pulse-X: example 1

```
int ssl_excert_prepend(SSL_EXCERT **pexc) {
    SSL_EXCERT *exc = app_malloc(sizeof(exc), “...”);
    memset(exc, 0, sizeof(*exc));
    ...
}
```

$[\mathrm{emp} \land \mathrm{true}]$ `app_malloc(sz, what)` $[\mathit{ok}\colon \mathrm{ret} \mapsto \mathrm{nil} \land \mathrm{true}]$

Pulse-X: example 1

```
int ssl_excert_prepend(SSL_EXCERT **pexc) {
    SSL_EXCERT *exc = app_malloc(sizeof(exc), "...");
    memset(exc, 0, sizeof(*exc));
    ...
}
```

$$[\mathrm{emp} \land \mathrm{true}]\ \texttt{app\_malloc(sz, what)}\ [ok: \mathrm{ret} \mapsto \mathrm{nil} \land \mathrm{true}]$$

Pulse-X: example 1

```
1. int ssl_excert_prepend(SSL_EXCERT **pexc) {
2.    SSL_EXCERT *exc = app_malloc(sizeof(exc), "...");
3.    memset(exc, 0, sizeof(*exc));
4.    ...
5. }
```

$[\mathsf{emp} \wedge \mathsf{true}]$ `app_malloc(sz, what)` $[ok\colon \mathsf{ret} \mapsto \mathsf{nil} \wedge \mathsf{true}]$

$[a \mapsto \mathsf{nil} \wedge \mathsf{true}]$ `memset(a,b,c)` $[er\colon a \mapsto \mathsf{nil} \wedge \mathsf{true}]$

Pulse-X: example 1

```
int ssl_excert_prepend(SSL_EXCERT **pexc) {
    SSL_EXCERT *exc = app_malloc(sizeof(exc), “...”);
    memset(exc, 0, sizeof(*exc));
    ...
}
```

$[\mathrm{emp} \land \mathrm{true}]$ `app_malloc(sz, what)` $[\mathit{ok}\colon \mathrm{ret} \mapsto \mathrm{nil} \land \mathrm{true}]$

$[a \mapsto \mathrm{nil} \land \mathrm{true}]$ `memset(a,b,c)` $[\mathit{er}\colon a \mapsto \mathrm{nil} \land \mathrm{true}]$

$[\mathrm{emp} \land \mathrm{true}]$ `ssl_excert_prepend(pexc)` $[\mathit{er}\colon \mathrm{emp} \land \mathrm{true}]$

Pulse-X: example 1

```
int ssl_excert_prepend(SSL_EXCERT **pexc) {
   SSL_EXCERT *exc = app_malloc(sizeof(exc), "...");
   memset(exc, 0, sizeof(*exc));
   ...
}
```

$[\mathsf{emp} \wedge \mathsf{true}]$ `app_malloc(sz, what)` $[\mathit{ok}\colon \mathsf{ret}\mapsto\mathsf{nil} \wedge \mathsf{true}]$

$[a\mapsto\mathsf{nil} \wedge \mathsf{true}]$ `memset(a,b,c)` $[\mathit{er}\colon a\mapsto\mathsf{nil} \wedge \mathsf{true}]$

$[\mathsf{emp} \wedge \mathsf{true}]$ `ssl_excert_prepend(pexc)` $[\mathit{er}\colon \mathsf{emp} \wedge \mathsf{true}]$

Manifest Bug: no matter the caller, it will always trigger the Bug

Pulse-X: example 2

Pulse-X: example 2

```
int chopup_args(ARGS *arg, char *buf, int *argc, char **argv[]) {
  int num, i;
  ...
  if (arg->count == 0) {
    arg->count = 20;
    arg->data = (char**)OPENSSL_malloc( sizeof(char*) * arg->count);
  }
 for (i=0; i<arg->count; i++)
   arg->data[i] = NULL;
...
}
```

Pulse-X: example 2

```
int chopup_args(ARGS *arg, char *buf, int *argc, char **argv[]) {
  int num, i;
  ...
  if (arg->count == 0) {
    arg->count = 20;
    arg->data = (char**)OPENSSL_malloc( sizeof(char*) * arg->count);
  }
 for (i=0; i<arg->count; i++)
   arg->data[i] = NULL;
...
}
```

Pulse-X: example 2

```
int chopup_args(ARGS *arg, char *buf, int *argc, char **argv[]) {
  int num, i;
  ...
  if (arg->count == 0) {
    arg->count = 20;
    arg->data = (char**)OPENSSL_malloc( sizeof(char*) * arg->count);
  }
 for (i=0; i<arg->count; i++)
   arg->data[i] = NULL;
...
}
```

Pulse-X: example 2

```
int chopup_args(ARGS *arg, char *buf, int *argc, char **argv[]) {
  int num, i;
  ...
  if (arg->count == 0) {
    arg->count = 20;
    arg->data = (char**)OPENSSL_malloc( sizeof(char*) * arg->count);
  }
 for (i=0; i<arg->count; i++)
   arg->data[i] = NULL;
 ...
 }
```

Pulse-X: example 2

```
1. int chopup_args(ARGS *arg, char *buf, int *argc, char **argv[]) {
2.   int num, i;
3.   ...
4.   if (arg->count == 0) {
5.     arg->count = 20;
6.     arg->data = (char**)OPENSSL_malloc( sizeof(char*) * arg->count);
7.   }
8.  for (i=0; i<arg->count; i++)
9.    arg->data[i] = NULL;
10. ...
11. }
```

Latent Bug: Bug triggered only if condition on line 4 is satisfied

Pulse-X: example 3

```
int main(int *argc, char **argv[]) {
  ...
  arg.count=0
  ...
  if( !chopop_args(&arg, buf, &argc,&argv)) break;
}
```

Pulse-X: example 3

```
int main(int *argc, char **argv[]) {
    ...
    arg.count=0
    ...
    if( !chopop_args(&arg, buf, &argc,&argv)) break;
}
```

Pulse-X: example 3

```
int main(int *argc, char **argv[]) {
    ...
    arg.count=0
    ...
    if( !chopop_args(&arg, buf, &argc,&argv)) break;
}
```

Pulse-X: example 3

```
int main(int *argc, char **argv[]) {
  ...
  arg.count=0
  ...
  if( !chopop_args(&arg, buf, &argc,&argv)) break;
}
```

Manifest Bug: Bug triggered regardless of the caller

Evaluation

Evaluation

Goal

Evaluation

Goal

- Pulse-X is superior in fix-rate

Evaluation

Goal

- Pulse-X is superior in fix-rate
- Pulse-X can find new bugs

Evaluation

Goal

- Pulse-X is superior in fix-rate
- Pulse-X can find new bugs

Target Program

Evaluation

Goal

- Pulse-X is superior in fix-rate
- Pulse-X can find new bugs

Target Program

- OpenSSL

Evaluation

Goal

- Pulse-X is superior in fix-rate
- Pulse-X can find new bugs

Target Program

- OpenSSL
  - < 8000 functions, < 400K lines of code

Evaluation

Goal

- Pulse-X is superior in fix-rate
- Pulse-X can find new bugs

Target Program

- OpenSSL
  - < 8000 functions, < 400K lines of code
- Correct Bug: only if it is fixed

Fix Rate

Fix Rate

Pulse-X

Fix Rate

Pulse-X

- Total: 26 bugs

Fix Rate

Pulse-X

- Total: 26 bugs
- Fixed: 19 bugs

Fix Rate

Pulse-X

- Total: 26 bugs
- Fixed: 19 bugs
- Fix rate: 73%

Fix Rate

Pulse-X

- Total: 26 bugs
- Fixed: 19 bugs
- Fix rate: 73%

Infer

Fix Rate

Pulse-X

- Total: 26 bugs
- Fixed: 19 bugs
- Fix rate: 73%

Infer

- Total: 80 bugs

Fix Rate

Pulse-X

- Total: 26 bugs
- Fixed: 19 bugs
- Fix rate: 73%

Infer

- Total: 80 bugs
- Fixed: 39 bugs

Fix Rate

Pulse-X

- Total: 26 bugs
- Fixed: 19 bugs
- Fix rate: 73%

Infer

- Total: 80 bugs
- Fixed: 39 bugs
- Fix rate: 50%

Fix Rate

Pulse-X

- Total: 26 bugs
- Fixed: 19 bugs
- Fix rate: 73%

Pulse-X has better fix rate!

Infer

- Total: 80 bugs
- Fixed: 39 bugs
- Fix rate: 50%

New Bugs

On recent version of OpenSSL

New Bugs

On recent version of OpenSSL

Pulse-X

New Bugs

On recent version of OpenSSL

Pulse-X

- Total: 30 bugs

New Bugs

On recent version of OpenSSL

Pulse-X

- Total: 30 bugs
- Fixed: 15 bugs

New Bugs

On recent version of OpenSSL

Pulse-X

- Total: 30 bugs
- Fixed: 15 bugs
- Pending: 5 bugs

New Bugs

On recent version of OpenSSL

Pulse-X

- Total: 30 bugs
- Fixed: 15 bugs
- Pending: 5 bugs
- Fix rate: 50%

New Bugs

On recent version of OpenSSL

Pulse-X

- Total: 30 bugs
- Fixed: 15 bugs
- Pending: 5 bugs
- Fix rate: 50%

Pulse-X can find new bugs!

Summary

Summary

Discrepancy as limitation

Summary

Discrepancy as limitation

- Theory: “Safe means Safe”

Summary

Discrepancy as limitation

- Theory: “Safe means Safe”
- Practice: “Bug means Bug”

Summary

Discrepancy as limitation

- Theory: “Safe means Safe”
- Practice: “Bug means Bug”

Solution: Pulse-X

Summary

Discrepancy as limitation

- Theory: “Safe means Safe”
- Practice: “Bug means Bug”

Solution: Pulse-X

- Incorrectness Separation Logic (ISL)

Summary

Discrepancy as limitation

- Theory: “Safe means Safe”
- Practice: “Bug means Bug”

Solution: Pulse-X

- Incorrectness Separation Logic (ISL)
    - Compositional analysis

Summary

Discrepancy as limitation

- Theory: “Safe means Safe”
- Practice: “Bug means Bug”

Solution: Pulse-X

- Incorrectness Separation Logic (ISL)
    - Compositional analysis
    - Under-approximated analysis

Summary

Discrepancy as limitation

- Theory: “Safe means Safe”
- Practice: “Bug means Bug”

Solution: Pulse-X

- Incorrectness Separation Logic (ISL)
  - Compositional analysis
  - Under-approximated analysis
- Better fix rate, Found new bugs