Under-Approximation for Scalable Bug Detection

Azalea Raad
Imperial College London

Iris Workshop
23 May 2023

azalea@imperial.ac.uk SoundAndComplete.org @azalearaad

State of the Art: ***Correctness***

- Lots of work on ***reasoning*** for proving ***correctness***
  - ➡ Prove the ***absence of bugs***
  - ➡ ***Over-approximate*** reasoning
  - ➡ ***Compositionality***

    in ***code*** ⇒ reasoning about ***incomplete components***

    in ***resources*** accessed ⇒ spatial locality
  - ➡ ***Scalability*** to large teams and codebases

Hoare Logic (HL)

Hoare triples $\{p\}$ C $\{q\}$ *iff* post(C)p $\subseteq$ q

For all states s *in* p
if running C *on* s *terminates in* s', *then* s' *is in* q

Hoare Logic (HL)

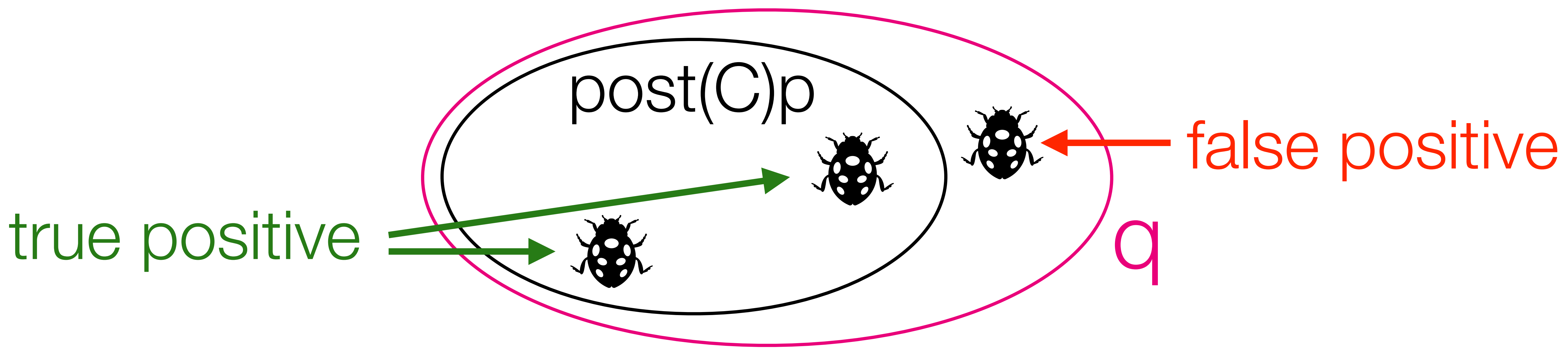
Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

q *over-approximates* post(C)p

Hoare Logic (HL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

q *over-approximates* post(C)p



“Don’t spam the developers!”

Incorrectness Logic:

A Formal Foundation

for

Bug Catching

Part I.

Incorrectness Logic (IL)

&

Incorrectness Separation Logic (ISL)

Incorrectness Logic (IL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

For all states s *in* p
if running C *on* s *terminates in* s', *then* s' *is in* q

Incorrectness Logic (IL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

For all states s *in* p
if running C *on* s *terminates in* s', *then* s' *is in* q

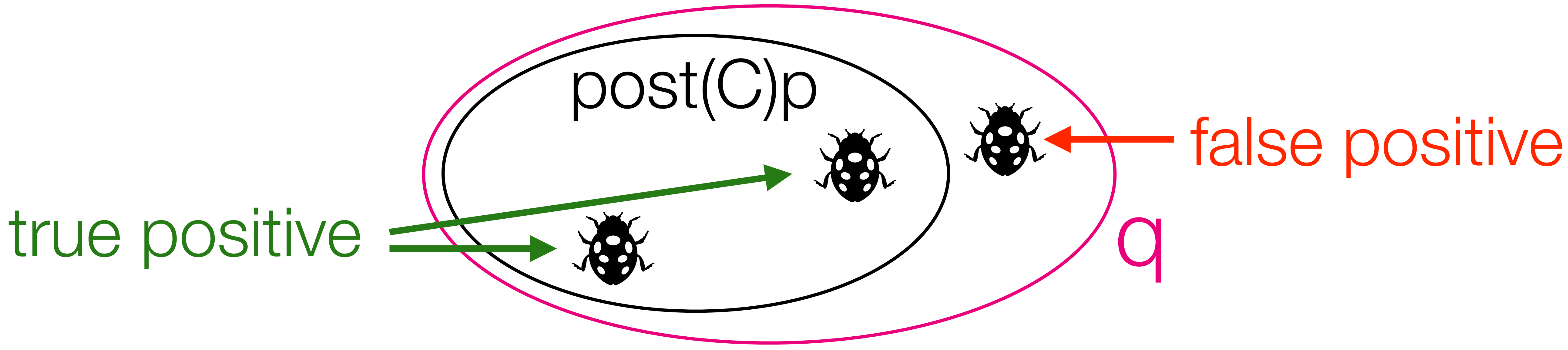
Incorrectness triples $[p]\ C\ [q]$ *iff* $post(C)p \supseteq q$

For all states s *in* q
s *can be reached by running* C *on some* s' *in* p

Incorrectness Logic (IL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

q *over-approximates* post(C)p



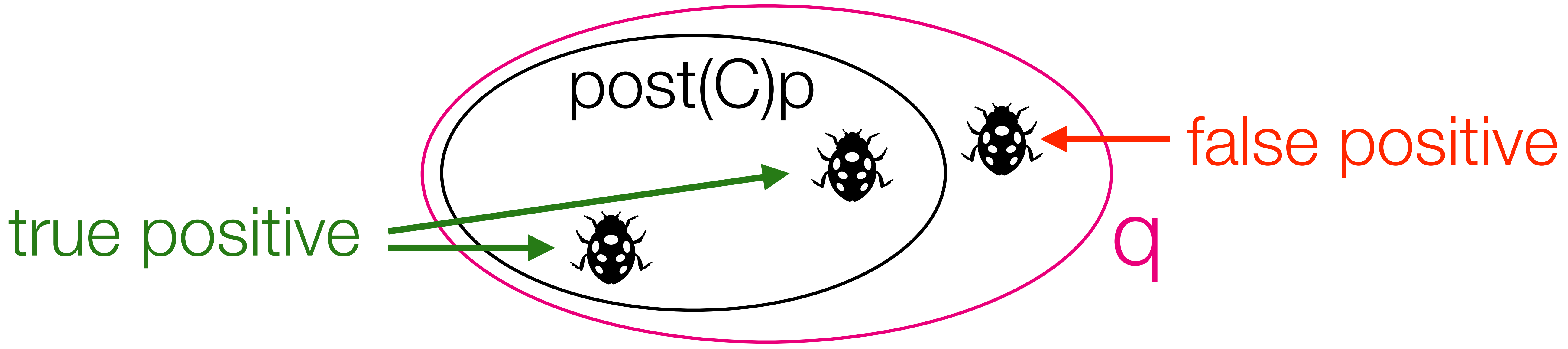
Incorrectness triples $[p]\ C\ [q]$ *iff* $post(C)p \supseteq q$

q *under-approximates* post(C)p

Incorrectness Logic (IL)

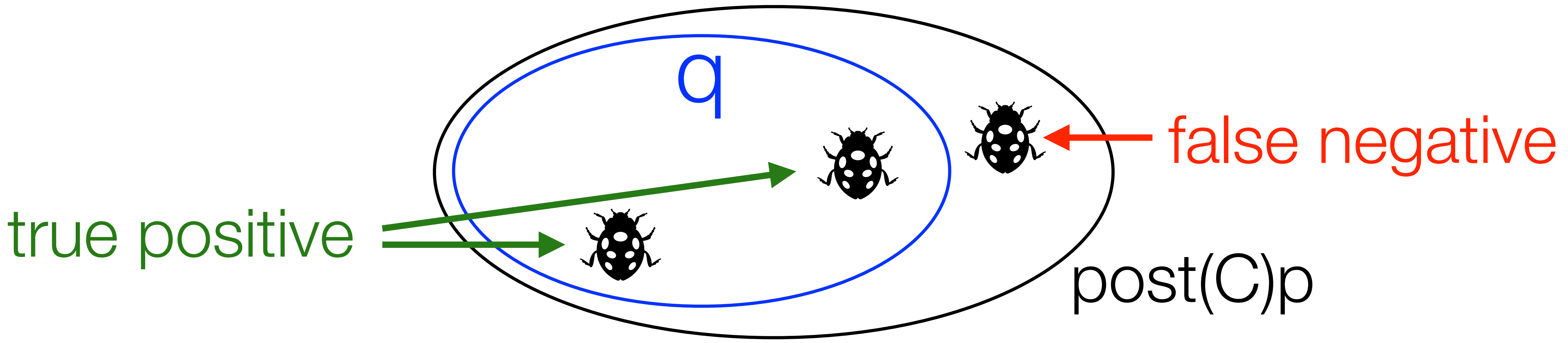
Hoare triples $\{p\}\ C\ \{q\}$ *iff* $\mathrm{post}(C)p \subseteq q$

q *over-approximates* post(C)p



Incorrectness triples $[p]\ C\ [q]$ *iff* $\mathrm{post}(C)p \supseteq q$

q *under-approximates* post(C)p



Incorrectness Logic (IL)

$$[p]\ C\ [\varepsilon\!: q]$$

ε: exit condition

ok: normal execution

er : erroneous execution

$[y{=}v]\ x{:=}y\ [ok\!: x{=}y{=}v]$

$[p]\ \mathrm{error}(\,)\ [er\!: p]$

Incorrectness Logic (IL)

$[p]\ C\ [\varepsilon\!:\ q]$ *iff* $\mathrm{post}(C, \varepsilon)p \supseteq q$

Incorrectness Logic (IL)

$\vdash_B$ [p] C [ε: q] *iff* post(C, ε)p $\supseteq$ q

Equivalent Definition (reachability)

$\vdash_B$ [p] C [ε: q] *iff* $\forall$ s $\in$ q. $\exists$ s' $\in$ p. (s',s) $\in$ [C]ε

IL Proof Rules and Principles (Sequencing)

$$\frac{[p]\ C_1\ [er: q]}{[p]\ C_1; C_2\ [er: q]}$$

- **Short-circuiting** semantics for errors

IL Proof Rules and Principles (Sequencing)

$$\frac{[p]\ C_1\ [er:\ q]}{[p]\ C_1;\ C_2\ [er:\ q]}$$

$$\frac{[p]\ C_1\ [ok:\ r] \qquad [r]\ C_2\ [\varepsilon:\ q]}{[p]\ C_1;\ C_2\ [\varepsilon:\ q]}$$

- **Short-circuiting** semantics for errors

IL Proof Rules and Principles (Branches)

$$\frac{[p]\ C_i\ [\varepsilon : q] \qquad \textbf{some}\ i \in \{1, 2\}}{[p]\ C_1 + C_2\ [\varepsilon : q]}$$

- **Drop paths/branches** (this is a **sound** under-approximation)
- **Scalable** bug detection!

$[p]\ C\ [\varepsilon : q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]_\varepsilon$

IL Proof Rules and Principles (Loops)

$$\frac{}{[p]\ C^*\ [ok: p]}\ \text{(Unroll-Zero)}$$

$$\frac{[p]\ C^*; C\ [\varepsilon: q]}{[p]\ C^*\ [\varepsilon: q]}\ \text{(Unroll-Many)}$$

- **Bounded unrolling of loops** (this is a **sound** under-approximation)
- **Scalable** bug detection!

$$[p]\ C\ [\varepsilon: q] \quad \textit{iff} \quad \forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$$

IL Proof Rules and Principles (Loops continued)

$$\frac{\forall n \in \mathbb{N}.\ [p(n)]\ C\ [ok: p(n+1)] \qquad k \in \mathbb{N}}{[p(0)]\ C^*\ [ok: p(k)]}\ \text{(Backwards-Variant)}$$

- Loop **invariants** are inherently **over-approximate**
- Reason about loops **under-approximately** via **sub-variants**

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall s \in q.\ \exists s' \in p.\ (s', s) \in [C]\varepsilon$

IL Proof Rules and Principles (Consequence)

$$\frac{p' \subseteq p \quad [p']\ C\ [\varepsilon: q'] \quad q' \supseteq q}{[p]\ C\ [\varepsilon: q]} \text{(Cons)}$$

- **Shrink** the **post** (e.g. drop disjuncts)
- **Scalable** bug detection!

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

IL Proof Rules and Principles (Consequence)

$$\frac{p' \subseteq p \quad [p']\ C\ [\varepsilon\!: q'] \quad q' \supseteq q}{[p]\ C\ [\varepsilon\!: q]}\ \text{(Cons)}$$

$$\frac{[p]\ C\ [\varepsilon\!: q_1 \vee q_2]}{[p]\ C\ [\varepsilon\!: q_1]}$$

- **Shrink** the **post** (e.g. drop disjuncts)
- **Scalable** bug detection!

$[p]\ C\ [\varepsilon\!: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

IL Proof Rules and Principles (Consequence)

$$\frac{p' \subseteq p \quad [p']\ C\ [\varepsilon: q'] \quad q' \supseteq q}{[p]\ C\ [\varepsilon: q]}\ \text{(Cons)}$$

$$\frac{p' \supseteq p \quad \{p'\}\ C\ \{q'\} \quad q' \subseteq q}{\{p\}\ C\ \{q\}}\ \text{(HL-Cons)}$$

$$\frac{[p]\ C\ [\varepsilon: q_1 \vee q_2]}{[p]\ C\ [\varepsilon: q_1]}$$

- **Shrink** the **post** (e.g. drop disjuncts)
- **Scalable** bug detection!

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

Incorrectness Logic: Summary

+ ***Under-approximate*** analogue of Hoare Logic

+ Formal foundation for ***bug catching***

– Global reasoning: ***non-compositional*** (as in original Hoare Logic)

– Cannot target ***memory safety bugs*** (e.g. use-after-free)

Incorrectness Logic: Summary

+ ***Under-approximate*** analogue of Hoare Logic

+ Formal foundation for ***bug catching***

– Global reasonin

– Cannot target *n*

<u>***Solution***</u>

Incorrectness Separation Logic

Incorrectness Separation Logic (ISL)

IL

$$[p]\ C\ [\varepsilon: q]$$

SL

$$\frac{\{p\}\ C\ \{q\}}{\{p * r\}\ C\ \{q * r\}}$$

$$x \mapsto - \; * \; x \mapsto - \;\Leftrightarrow\; \text{false}$$

$$x \mapsto v \; * \; \text{emp} \;\Leftrightarrow\; x \mapsto v$$

ISL

$$\frac{[p]\ C\ [\varepsilon: q]}{[p * r]\ C\ [\varepsilon: q * r]}$$

$$x \mapsto v \; * \; x \mapsto v' \;\Leftrightarrow\; \text{false}$$

$$x \mapsto v \; * \; \text{emp} \;\Leftrightarrow\; x \mapsto v$$

ISL: Local Axioms

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto]$

$[x = null]$ free(x) $[er: x = null]$

FREE

$[x \not\mapsto]$ free(x) $[er: x \not\mapsto]$

null-pointer-dereference error

double-free error

ISL: Local Axioms

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto]$

FREE

$[x{=}null]$ free(x) $[er: x{=}null]$

$[x \not\mapsto]$ free(x) $[er: x \not\mapsto]$

null-pointer-dereference error

double-free error

$[x \mapsto v']$ [x]:= v $[ok: x \mapsto v]$

WRITE

$[x{=}null]$ [x]:= v $[er: x{=}null]$

$[x \not\mapsto]$ [x]:= v $[er: x \not\mapsto]$

ISL: Local Axioms

$[x \mapsto v]\ \mathrm{free}(x)\ [ok{:}\ x \not\mapsto]$

$[x{=}null]\ \mathrm{free}(x)\ [er{:}\ x{=}null]$

$[x \not\mapsto]\ \mathrm{free}(x)\ [er{:}\ x \not\mapsto]$

FREE

null-pointer-dereference error

double-free error

$[x \mapsto v']\ [x]{:=}\ v\ [ok{:}\ x \mapsto v]$

$[x{=}null]\ [x]{:=}\ v\ [er{:}\ x{=}null]$

$[x \not\mapsto]\ [x]{:=}\ v\ [er{:}\ x \not\mapsto]$

WRITE

$[x \mapsto v]\ y{:=}\ [x]\ [ok{:}\ x \mapsto v \wedge y{=}v]$

$[x{=}null]\ y{:=}\ [x]\ [er{:}\ x{=}null]$

$[x \not\mapsto]\ y{:=}\ [x]\ [er{:}\ x \not\mapsto]$

READ

ISL: Local Axioms

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto]$

$[x = null]$ free(x) $[er: x = null]$

$[x \not\mapsto]$ free(x) $[er: x \not\mapsto]$

FREE

null-pointer-dereference error

double-free error

$[x \mapsto v']$ [x]:= v $[ok: x \mapsto v]$

$[x = null]$ [x]:= v $[er: x = null]$

$[x \not\mapsto]$ [x]:= v $[er: x \not\mapsto]$

WRITE

$[x \mapsto v]$ y:= [x] $[ok: x \mapsto v \wedge y = v]$

$[x = null]$ y:= [x] $[er: x = null]$

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

READ

$[emp]$ x:= alloc() $[ok: \exists l.\ l \mapsto v \wedge x = l]$

ALLOC

ISL Summary

- Incorrectness ***Separation*** Logic (ISL)
  - ➡ IL + SL for ***compositional bug catching***
  - ➡ ***Under-approximate*** analogue of SL
  - ➡ Targets ***memory safety bugs*** (e.g. use-after-free)
- Combining IL+SL: not straightforward
  - ➡ ***invalid frame*** rule!
- Fix: a ***monotonic model*** for frame preservation
- Recovering the ***footprint property*** for completeness
- ISL-based ***analysis***
  - ➡ ***No-false-positives theorem:***

    All bugs found are true bugs

Part II.

Pulse-X: ISL for Scalable Bug Detection

Pulse-X at a Glance

- **Automated** program analysis for **memory safety errors** (NPEs, UAFs) and **leaks**
- Underpinned by ISL (under-approximate) — **no false positives***
- **Inter-procedural** and **bi-abductive** — under-approximate analogue of Infer
- **Compositional** (begin-anywhere analysis) — important for CI
- Deployed at Meta
- **Performance**: comparable to Infer, though merely an academic tool!
- **Fix rate**: comparable or better than Infer!
- Three dimensional scalability
  - code size (large codebases)
  - people (large teams, CI)
  - speed (high frequency of code changes)

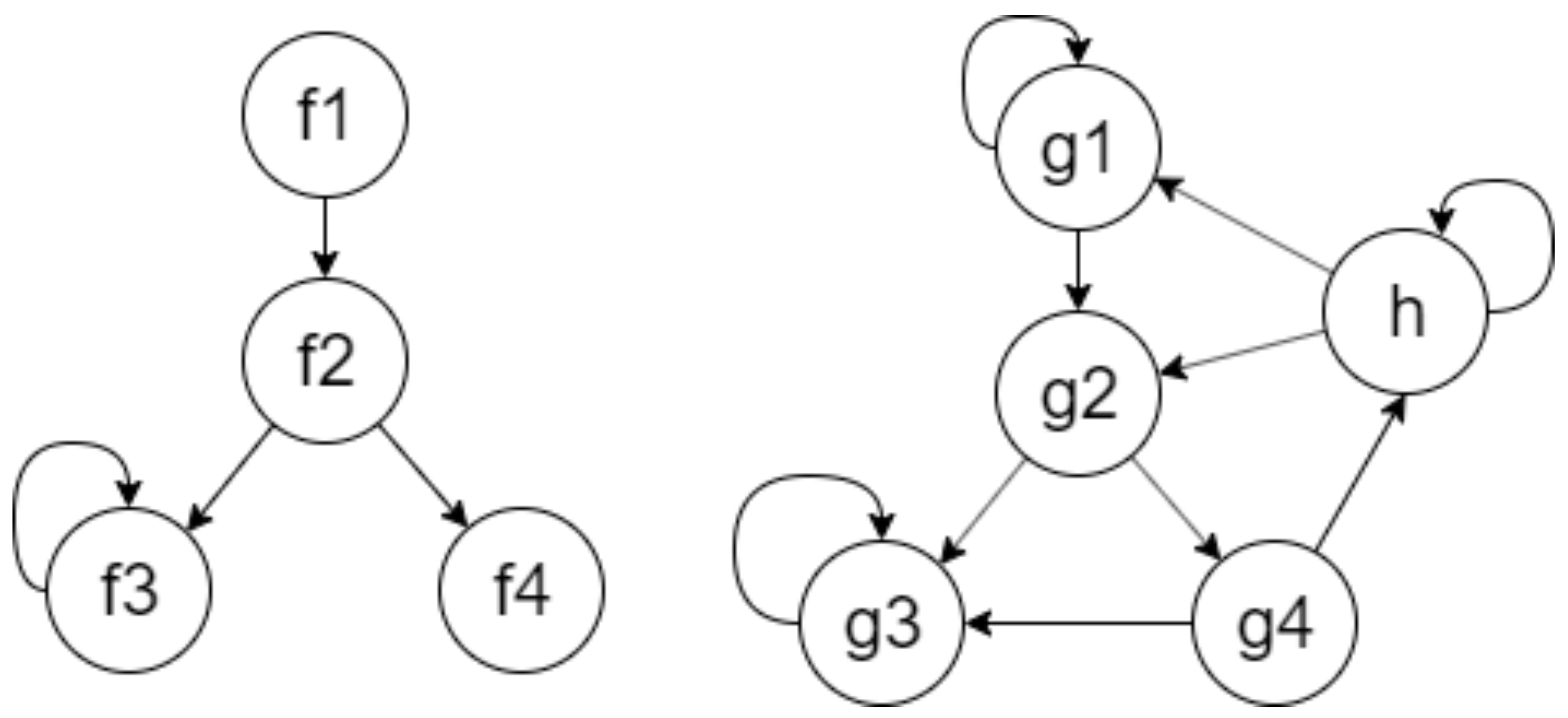
Compositional, Begin-Anywhere Analysis

- Analysis result of a program = analysis results of its parts
  +
  a method of combining them

Compositional, Begin-Anywhere Analysis

- Analysis result of a program = analysis results of its parts
  +
  a method of combining them

  - ➡ Parts: Procedures



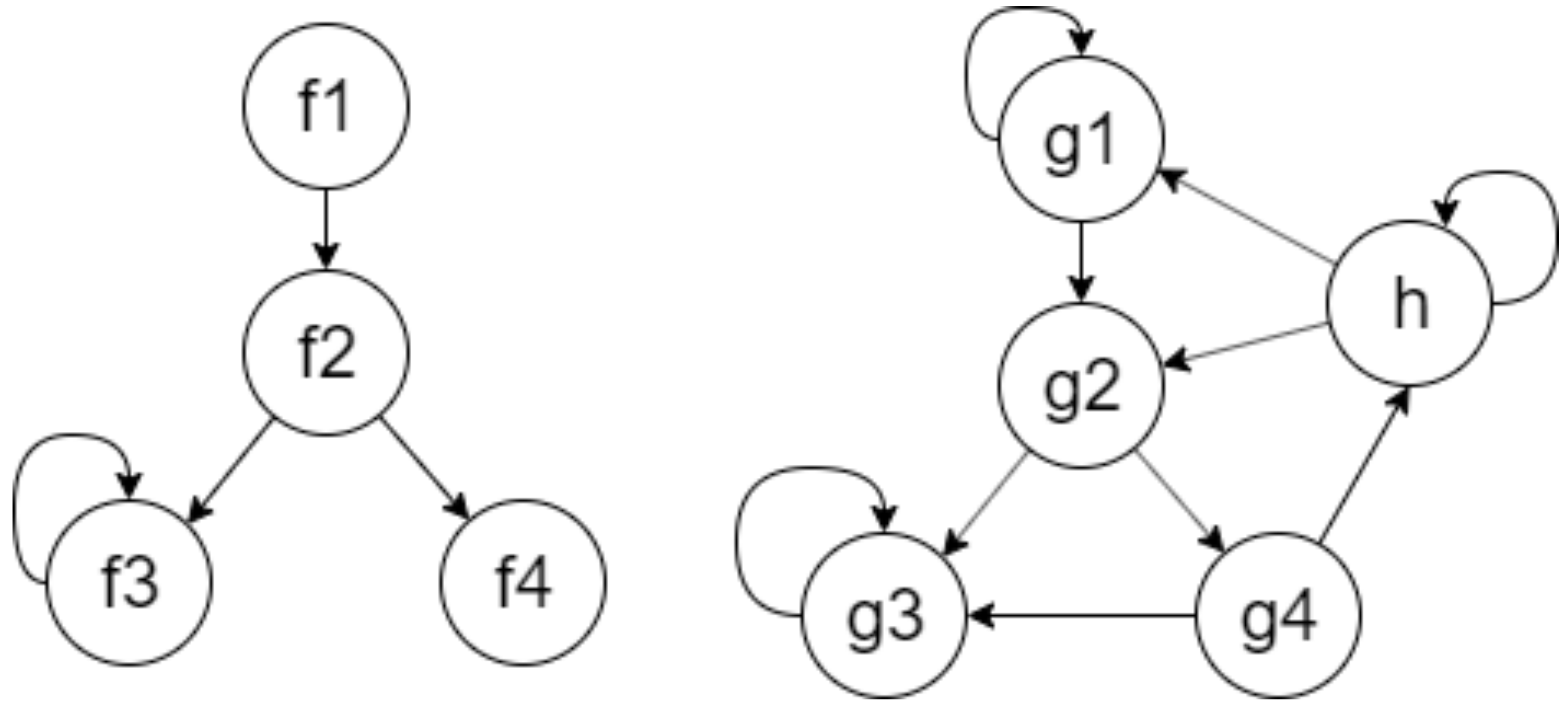
Compositional, Begin-Anywhere Analysis

❖ Analysis result of a program = analysis results of its parts

\+

a method of combining them

➡ Parts: Procedures

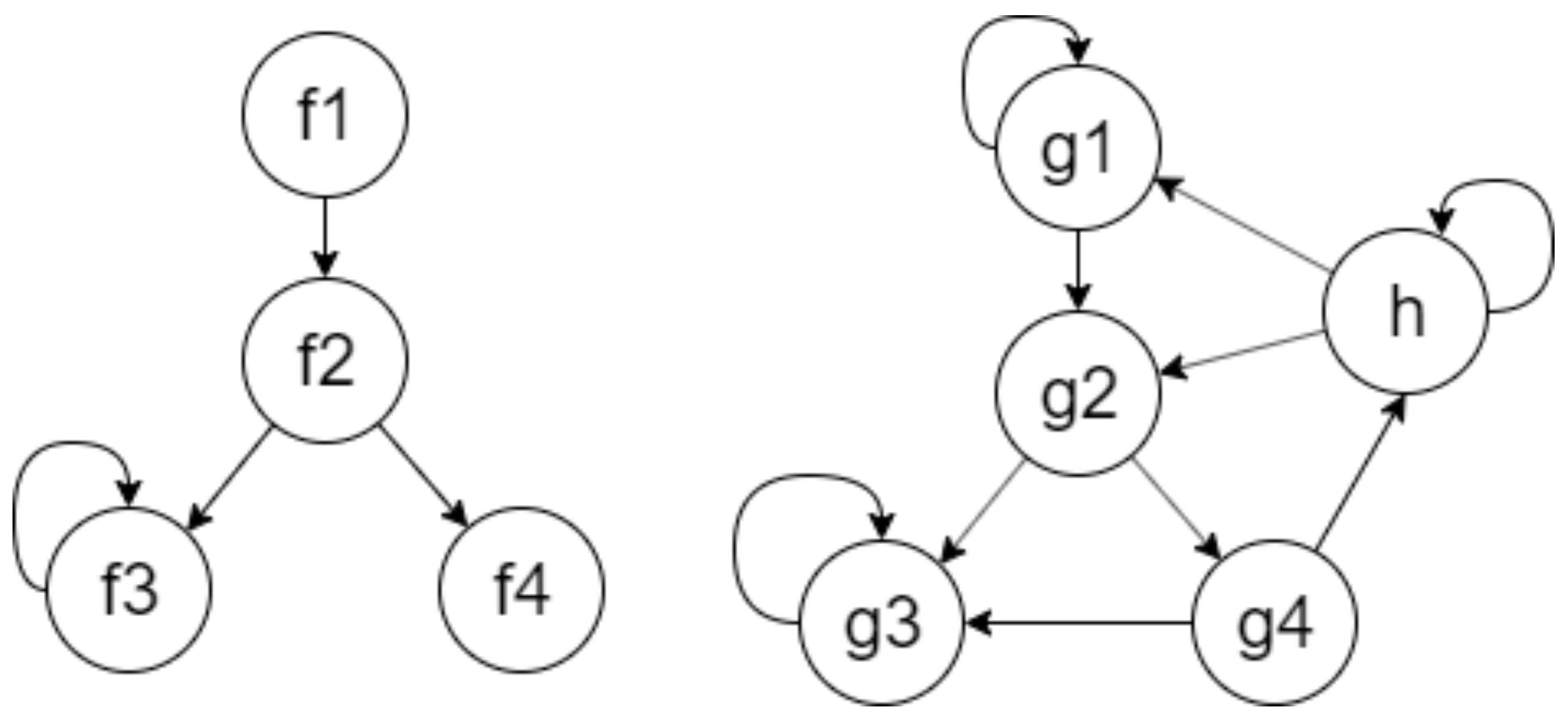


➡ Method: under-approximate bi-abduction

Compositional, Begin-Anywhere Analysis

❖ Analysis result of a program = analysis results of its parts
+
a method of combining them

➡ Parts: Procedures



➡ Method: under-approximate bi-abduction

➡ Analysis result: incorrectness triples (under-approximate specs)

Pulse-X Algorithm: Proof Search in ISL

- Analyse each procedure f in isolation, find its **summary** (collection of ISL triples)
  - ➡ A **summary table** T, initially populated only with local (pre-defined) axioms
  - ➡ Use bi-abduction and T to find the summary of f
  - ➡ Recursion: bounded unrolling
  - ➡ Extend T with the summary of f
- Similar bi-abductive mechanism to Infer, but:
  - ➡ Can **soundly** drop execution paths/branches
  - ➡ Can **soundly** bound loop unrolling

Pulse-X: Null Pointer Dereference in OpenSSL

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

calls CRYPTO_malloc (a malloc wrapper)

Pulse-X: Null Pointer Dereference in OpenSSL

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

calls CRYPTO_malloc (a malloc wrapper)

null pointer dereference

CRYPTO_malloc may return null!

Pulse-X: Null Pointer Dereference in OpenSSL

```
int ssl_excert_prepend(...){
    SSL_EXCERT *exc= app_malloc(sizeof(*exc), "prepend cert”);
    memset(exc, 0, sizeof(*exc));
    …
}
```

calls CRYPTO_malloc (a malloc wrapper)

null pointer dereference

CRYPTO_malloc may return null!

[emp] *exc= app_malloc(sz, …) [ok: exc = null]

+

[exc = null] memset(exc,-,-) [er: exc = null]

[emp] ssl_excert_prepend(…) [er: exc = null]

Pulse-X: Null Pointer Dereference in OpenSSL

apps/lib/s_cb.c Outdated — Hide resolved

```
@@ -956,6 +956,9 @@ static int ssl_excert_prepend(SSL_EXCERT **pexc)
 {
     SSL_EXCERT *exc = app_malloc(sizeof(*exc), "prepend cert");

+    if (!exc) {
```

paulidale 13 days ago Contributor

False positive, `app_malloc()` doesn't return if the allocation fails.

lequangloc 13 days ago Author

Our tool recognizes app_malloc() in test/testutil/apps_mem.c rather than the one in apps/lib/apps.c. While the former doesn't return if the allocation fails, the latter does. How do we know which one is actually called?

paulidale 13 days ago Contributor

It would need to look at the link lines or build dependencies to figure out which sources were used.

We should fix the one in `test/testutil/apps_mem.c`.

Pulse-X: Null Pointer Dereference in OpenSSL

apps/lib/s_cb.c Outdated   Hide resolved

```
...   ...   @@ -956,6 +956,9 @@ static int ssl_excert_prepend(SSL_EXCERT **pexc)
956   956    {
957   957        SSL_EXCERT *exc = app_malloc(sizeof(*exc), "prepend cert");
958   958
      959   +        if (!exc) {
```

paulidale 13 days ago Contributor

False positive, `app_malloc()` doesn't return if the allocation fails.

lequangloc 13 days ago Author

Our tool recognizes app_malloc() in test/testutil/apps_mem.c rather than the one in apps/lib/apps.c. While the former doesn't return if the allocation fails, the latter does. How do we know which one is actually called?

paulidale 13 days ago Contributor

It would need to look at the link lines or build dependencies to figure out which sources were used.

We should fix the one in `test/testutil/apps_mem.c` .

Created pull request #15836 to commit the fix.

Pulse-X: Bug Reporting

No False Positives: Report ***All*** Bugs Found?

Not quite…

Pulse-X: Bug Reporting

```
1.void foo(int *x){
2.    *x = 42;
}
```

WRITE [x=null] *x = v [er: x=null]

[x=null] foo(x) [er: x=null]

Should we report this NPD?

Pulse-X: Bug Reporting

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

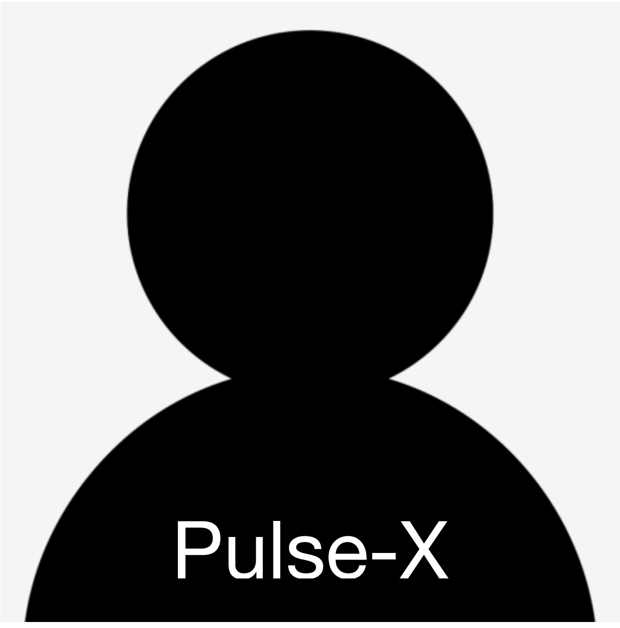
WRITE [x=null] *x = v [er: x=null]

[x=null] foo(x) [er: x=null]

Should we report this NPD?

yes **no**

“But I never call foo with null!”



“Which bugs shall I report then?”

Pulse-X: Bug Reporting

Problem

Must consider the **whole program**
to decide whether to report

Solution

Manifest Errors

Pulse-X: ***Manifest*** Errors

- Intuitively: the error occurs for **all input states**
- Formally: [p] C [er: q] is manifest iff:

$$\forall\, s.\ \exists\, s'.\ \ (s,s') \in [C]_{er}\ \wedge\ s' \in (q * \mathrm{true})$$

- Algorithmically: …

Pulse-X: Null Pointer Dereference in OpenSSL

```
1.int ssl_excert_prepend(...){
2.    SSL_EXCERT *exc= app_malloc(sizeof(*exc), "prepend cert”);
3.    memset(exc, 0, sizeof(*exc));
      …
}
```

calls CRYPTO_malloc (a malloc wrapper)

null pointer dereference

CRYPTO_malloc may return null!

[emp] `ssl_excert_prepend(…)` [er: exc = null]

Pulse-X: Null Pointer Dereference in OpenSSL

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

calls CRYPTO_malloc (a malloc wrapper)

null pointer dereference

CRYPTO_malloc may return null!

[emp] `ssl_excert_prepend(…)` [er: exc = null]

Manifest Error (all calls to `ssl_excert_prepend` can trigger the error)!

Pulse-X: ***Latent*** Errors

An error triple [p] C [er: q] is <u>latent</u> iff it is not manifest

Pulse-X: Latent Error

```
int chopup_args(ARGS *args,…){
    …
    if (args->count == 0 ) {
      args->count=20;
      args->data= (char**)ssl_excert_prepend(…);
    }
    for (i=0; i<args->count; i++) {
      args->data[i]=NULL;
      …
    }
```

Pulse-X: Latent Error

```c
int chopup_args(ARGS *args,…){
    …
    if (args->count == 0 ) {
      args->count=20;
      args->data= (char**)ssl_excert_prepend(…);
    }
    for (i=0; i<args->count; i++) {
      args->data[i]=NULL;
      …
    }
```

null pointer
dereference

Pulse-X: Latent Error

```
int chopup_args(ARGS *args,…){
    …
    if (args->count == 0 ) {
      args->count=20;
      args->data= (char**)ssl_excert_prepend(…);
    }
    for (i=0; i<args->count; i++) {
      args->data[i]=NULL;
      …
    }
```

null pointer dereference

Latent Error:

only calls with `args->count==0` can trigger the error

Pulse-X: Memory Leak in OpenSSL

```
static int www_body(…){
  ...
  io = BIO_new(BIO_f_buffer());
  ssl_bio BIO_new(BIO_f_ssl());
  ...
  BIO_push(io, ssl_bio);
  ...
  BIO_free_all(io);
  ...
  return ret;
}
```

Pulse-X: Memory Leak in OpenSSL

```
static int www_body(…){
  ...
  io = BIO_new(BIO_f_buffer());
  ssl_bio BIO_new(BIO_f_ssl());
  ...
  BIO_push(io, ssl_bio);
  ...
  BIO_free_all(io);
  ...
  return ret;
}
```

does nothing when `io` is null

Pulse-X: Memory Leak in OpenSSL

```
static int www_body(…){
  ...
  io = BIO_new(BIO_f_buffer());
  ssl_bio BIO_new(BIO_f_ssl());
  ...
  BIO_push(io, ssl_bio);
  ...
  BIO_free_all(io);
  ...
  return ret;
}
```

does nothing when `io` is null

leaks `ssl_bio`

Pulse-X: Memory Leak in OpenSSL

```
static int www_body(…){
   ...
   io = BIO_new(BIO_f_buffer());
   ssl_bio BIO_new(BIO_f_ssl());
   ...
   BIO_push(io, ssl_bio);
   ...
   BIO_free_all(io);
   ...
   return ret;
}
```

does nothing when `io` is null

leaks `ssl_bio`

426 lines of complex code:
`io` manipulated by several procedures and multiple loops

Pulse-X performs under-approximation with bounded loop unrolling

No-False Positives: Caveat

- ❖ Unknown procedures (e.g. where the code is unavailable) are treated as `skip`
- ❖ Incomplete arithmetic solver

Speed (fast but simplistic)	vs	Precision (slow but accurate)

"Scientists seek perfection and are idealists. ... An engineer's task is to not be idealistic. You need to be realistic as you have to compromise between conflicting interests."

Pulse-X Summary

- ➡ Automated program analysis for detecting memory safety errors and leaks
- ➡ Manifest errors (underpinned by ISL): no false positives*
- ➡ compositional, scalable, begin-anywhere

Part III.

ISL Extensions:

Concurrent Incorrectness Separation Logic (CISL)

&

Concurrent Adversarial Separation Logic (CASL)

&

Incorrectness Non-Termination Logic (INTL)

Termination vs Non-Termination

- Showing **termination** is compatible with **correctness** frameworks:
  - ➡ **Every** trace of a given program must terminate
  - ➡ Inherently **over-approximate**

```
skip + x:=1
```

Termination vs Non-Termination

- Showing **termination** is compatible with **correctness** frameworks:
  - ➡ **Every** trace of a given program must terminate
  - ➡ Inherently **over-approximate**

```
skip + x:=1
```

- Showing **non-termination** compatible with **incorrectness** frameworks:
  - ➡ **Some** trace of a given program must not-terminate
  - ➡ Inherently **under-approximate**

```
skip + while(true)skip
```

Incorrectness Non-Termination Logic (INTL)

- A framework for **detecting non-termination bugs**
- Supports **unstructured** constructs (goto), as well exceptions and breaks
- Reasons for non-termination:
  - ➡ Infinite loops
  - ➡ Infinite recursion
  - ➡ Cyclic `goto` soups

INTL Divergence Proof Rules

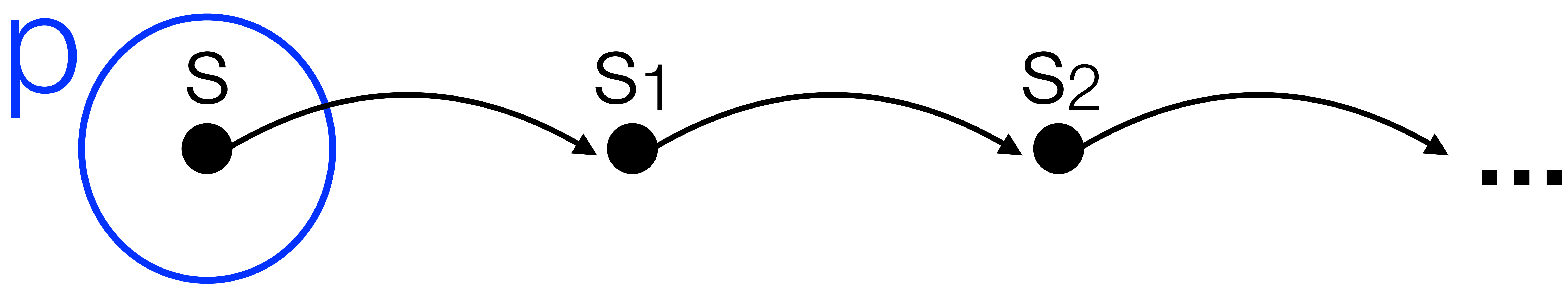
$$[p]\ C\ [\infty]$$

C has divergent traces starting from p

INTL Divergence Proof Rules

$$[p]\ C\ [\infty]$$

C has divergent traces starting from p



INTL Divergence Proof Rules (Sequencing)

$$\frac{[p]\ C_1\ [\infty]}{[p]\ C_1;\ C_2\ [\infty]}$$

INTL Proof Rules and Principles

INTL Proof Rules

=

(Under-Approximate) IL/ISL Proof Rules

+

Divergence (Non-Termination) Rules

INTL Divergence Proof Rules (Sequencing)

$$\frac{[p]\ C_1\ [\infty]}{[p]\ C_1;\ C_2\ [\infty]}$$

$$\frac{\vdash_B [p]\ C_1\ [ok\!:\ q] \qquad [q]\ C_2\ [\infty]}{[p]\ C_1;\ C_2\ [\infty]}$$

INTL Divergence Proof Rules (Branches)

$$\frac{[p]\ C_i\ [\infty] \qquad\qquad \textbf{some } i \in \{1, 2\}}{[p]\ C_1 + C_2\ [\infty]}$$

- **Drop paths/branches** (this is a **sound** under-approximation)
- **Scalable** bug detection!

INTL Divergence Proof Rules (Loops — first attempt)

$$\frac{[q]\ C;\ C^{*}\ [\infty]}{[p]\ C^{*}\ [\infty]}$$

INTL Divergence Proof Rules (Loops — first attempt)

$$\frac{[q]\ C;\ C^{*}\ [\infty]}{[p]\ C^{*}\ [\infty]}$$

$$\frac{[p]\ C_1\ [\infty]}{[p]\ C_1;\ C_2\ [\infty]}$$

INTL Divergence Proof Rules (Loops — first attempt)

$$\frac{[q]\ C;\ C^{*}\ [\infty]}{[p]\ C^{*}\ [\infty]}$$

$$\frac{[p]\ C\ [\infty]}{[p]\ C^{*}\ [\infty]}\ \text{(derived)}$$

$$\frac{[p]\ C_1\ [\infty]}{[p]\ C_1;\ C_2\ [\infty]}$$

INTL Divergence Proof Rules (Loops — first attempt)

$$\frac{[q]\ C;\ C^*\ [\infty]}{[p]\ C^*\ [\infty]}$$

$$\frac{[p]\ C\ [\infty]}{[p]\ C^*\ [\infty]}\ \text{(derived)}$$

$$\frac{\vdash_B [p]\ C\ [ok: p]}{[p]\ C^*\ [\infty]}$$

INTL Divergence Proof Rules (Loops — first attempt)

$$\frac{[q]\ C;\ C^*\ [\infty]}{[p]\ C^*\ [\infty]}$$

$$\frac{[p]\ C\ [\infty]}{[p]\ C^*\ [\infty]}\ \text{(derived)}$$

$$\frac{\vdash_B [p]\ C\ [ok:\ p]}{[p]\ C^*\ [\infty]}$$

INTL Divergence Proof Rules (While Loops — first attempt)

$$[p \land b] \text{ while(b) } C \ [\infty]$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

INTL Divergence Proof Rules (While Loops — first attempt)

$$\frac{[p \wedge b]\ (\text{assume}(b);\ C)^*;\ \text{assume}(!b)\ [\infty]}{[p \wedge b]\ \text{while}(b)\ C\ [\infty]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

INTL Divergence Proof Rules (While Loops — first attempt)

$$\frac{[p \wedge b]\ (\text{assume}(b);\ C)^*;\ \text{assume}(!b)\ [\infty]}{[p \wedge b]\ \text{while}(b)\ C\ [\infty]}$$

$$\frac{[p]\ C_1\ [\infty]}{[p]\ C_1;\ C_2\ [\infty]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

INTL Divergence Proof Rules (While Loops — first attempt)

$$\dfrac{\dfrac{[p \wedge b]\ (\mathrm{assume}(b);\ C)^*;[\infty]}{[p \wedge b]\ (\mathrm{assume}(b);\ C)^*;\ \mathrm{assume}(!b)\ [\infty]}}{[p \wedge b]\ \mathrm{while}(b)\ C\ [\infty]}$$

$$\dfrac{[p]\ C_1\ [\infty]}{[p]\ C_1;\ C_2\ [\infty]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

INTL Divergence Proof Rules (While Loops — first attempt)

$$\frac{\vdash_B [p]\ C\ [ok: p]}{[p]\ C^*\ [\infty]}$$

$$\cfrac{\cfrac{[p \land b]\ (\mathrm{assume}(b);\ C)^*;[\infty]}{[p \land b]\ (\mathrm{assume}(b);\ C)^*;\ \mathrm{assume}(!b)\ [\infty]}}{[p \land b]\ \mathrm{while}(b)\ C\ [\infty]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

INTL Divergence Proof Rules (While Loops — first attempt)

$$\dfrac{\dfrac{\dfrac{\vdash_B [p \wedge b]\ \text{assume}(b);\ C\ [ok: p \wedge b]}{[p \wedge b]\ (\text{assume}(b);\ C)^*;[\infty]}}{[p \wedge b]\ (\text{assume}(b);\ C)^*;\ \text{assume}(!b)\ [\infty]}}{[p \wedge b]\ \text{while}(b)\ C\ [\infty]}$$

$$\dfrac{\vdash_B [p]\ C\ [ok: p]}{[p]\ C^*\ [\infty]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

INTL Divergence Proof Rules (While Loops — first attempt)

$$\dfrac{\dfrac{\dfrac{\vdash_B [p \wedge b]\ \mathrm{assume}(b);\ C\ [ok: p \wedge b]}{[p \wedge b]\ (\mathrm{assume}(b);\ C)^{*};[\infty]}}{[p \wedge b]\ (\mathrm{assume}(b);\ C)^{*};\ \mathrm{assume}(!b)\ [\infty]}}{[p \wedge b]\ \mathrm{while}(b)\ C\ [\infty]}$$

$$\vdash_B\ [p \wedge b]\ \mathrm{assume}(b)\ [ok: p \wedge b]$$

$$\dfrac{\vdash_B [p]\ C_1\ [ok: r] \qquad \vdash_B [r]\ C_2\ [\varepsilon: q]}{[p]\ C_1;\ C_2\ [\varepsilon: q]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

INTL Divergence Proof Rules (While Loops — first attempt)

$$\dfrac{\dfrac{\dfrac{\vdash_B [p \wedge b]\ C\ [ok: p \wedge b]}{\vdash_B [p \wedge b]\ \mathrm{assume}(b);\ C\ [ok: p \wedge b]}}{[p \wedge b]\ (\mathrm{assume}(b);\ C)^*;[\infty]}}{\dfrac{[p \wedge b]\ (\mathrm{assume}(b);\ C)^*;\ \mathrm{assume}(!b)\ [\infty]}{[p \wedge b]\ \mathrm{while}(b)\ C\ [\infty]}}$$

$\vdash_B\ [p \wedge b]$ assume(b) $[ok: p \wedge b]$

$$\dfrac{\vdash_B [p]\ C_1\ [ok: r] \qquad \vdash_B [r]\ C_2\ [\varepsilon: q]}{[p]\ C_1;\ C_2\ [\varepsilon: q]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

INTL Divergence Proof Rules (While Loops — first attempt)

$$\frac{\vdash_B [p \land b]\ C\ [ok: p \land b]}{[p \land b]\ \text{while}(b)\ C\ [\infty]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

INTL Divergence Proof Rules (While Loops — first attempt)

$$\frac{\vdash_B [p \wedge b]\ C\ [ok: p \wedge b]}{[p \wedge b]\ \text{while}(b)\ C\ [\infty]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

INTL Divergence Proof Rules (Loops — first attempt)

Program `while(x > 0) x--` always terminates. But…

$$\frac{\vdash_B [p \land b]\ C\ [ok: p \land b]}{[p]\ \text{while}(b)\ C\ [\infty]}$$

INTL Divergence Proof Rules (Loops — first attempt)

Program `while(x > 0) x--` always terminates. But…

$$[\, x > 0 \,] \;\; \texttt{while(x > 0) x--} \;\; [\infty]$$

$$\frac{\vdash_B [p \wedge b]\; C\; [ok: p \wedge b]}{[p]\; \texttt{while}(b)\; C\; [\infty]}$$

INTL Divergence Proof Rules (Loops — first attempt)

Program `while(x > 0) x--` always terminates. But…

$$\frac{\vdash_B [x > 0]\ \texttt{x--}\ [ok\!: x > 0]}{[\ x > 0\ ]\ \ \texttt{while(x > 0) x--}\ \ [\infty]}$$

$$\frac{\vdash_B [p \wedge b]\ C\ [ok\!: p \wedge b]}{[p]\ \texttt{while}(b)\ C\ [\infty]}$$

INTL Divergence Proof Rules (Loops — first attempt)

Program `while(x > 0) x--` always terminates. But…

$$\frac{\vdash_B [x > 0]\ \texttt{x--}\ [ok: x > 0]}{[\ x > 0\ ]\ \ \texttt{while(x > 0) x--}\ \ [\infty]}$$

$$\frac{\vdash_B [p \land b]\ C\ [ok: p \land b]}{[p]\ \texttt{while(b)}\ C\ [\infty]}$$

$\vdash_B [p]\ C\ [\varepsilon: q]$

iff

$\forall\ s \in q.\ \exists\ s' \in p.\ (s', s) \in [C]\varepsilon$

Problem

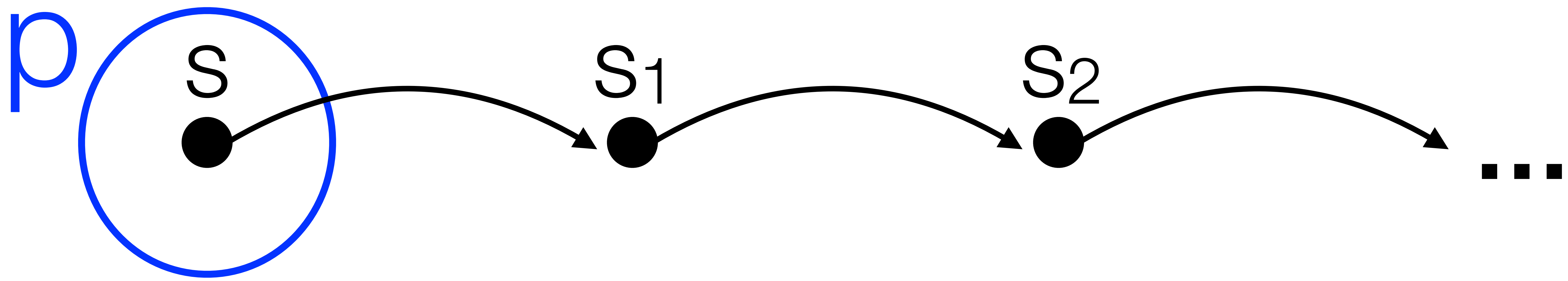
- Premise: p reached by executing C on some p
- I.e. in the **backward** direction
- Can construct a *backward infinite* trace

$$\frac{\vdash_B [p]\ C\ [ok: p]}{[p]\ C^*\ [\infty]}$$

Problem

- Premise: p reached by executing C on some p
- I.e. in the **backward** direction
- Can construct a *backward infinite* trace
- We need a *forward infinite* trace

$$\frac{\vdash_B [p]\ C\ [ok\!:\ p]}{[p]\ C^*\ [\infty]}$$



Problem

- Premise: p reached by executing C on some p
- I.e. in the **backward** direction
- Can constru
- We need a f

Solution

Forward *Under-Approximate Triples*

Forward Under-Approximate (FUX) Triples

$$\vdash_F [p]\ C\ [\varepsilon: q] \quad \textit{iff} \quad \forall s \in p.\ \exists s' \in q.\ (s,s') \in [C]\varepsilon$$

Forward Under-Approximate (FUX) Triples

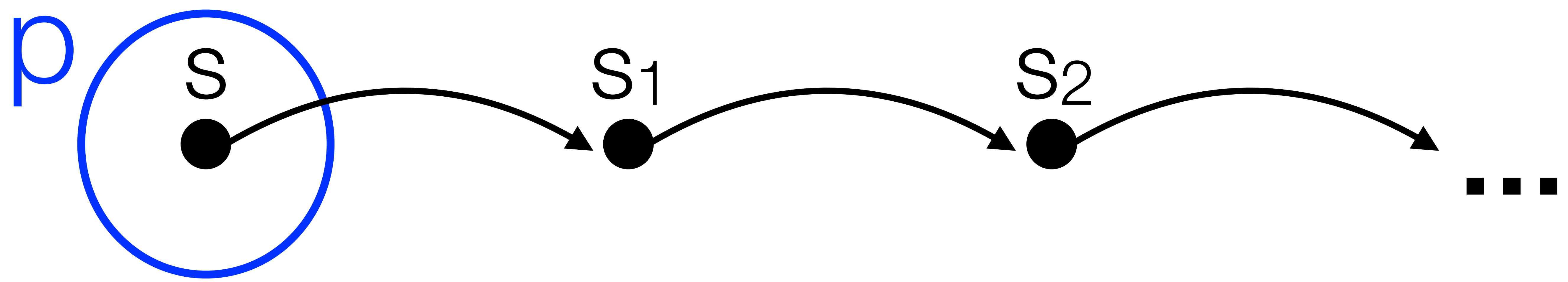
$\vdash_F [p]\ C\ [\varepsilon\!: q]$ *iff* $\forall\, s \in p.\ \exists\, s' \in q.\ (s,s') \in [C]\varepsilon$

$$\frac{\vdash_F [p]\ C\ [ok\!: p]}{[p]\ C^*\ [\infty]}$$

Forward Under-Approximate (FUX) Triples

$\vdash_F [p]\ C\ [\varepsilon: q]$ *iff* $\forall s \in p.\ \exists s' \in q.\ (s,s') \in [C]\varepsilon$

$$\frac{\vdash_F [p]\ C\ [ok: p]}{[p]\ C^*\ [\infty]}$$



FUX is **Under-Approximate**!

$\vdash_F$ $[p]\ C\ [\varepsilon\!: q]$ *iff* $\forall\, s \in p.\ \exists\, s' \in q.\ (s,s') \in [C]\varepsilon$

FUX is **Under-Approximate**!

$$\vdash_F [p]\ C\ [\varepsilon: q] \quad \textit{iff} \quad \forall s \in p.\ \exists s' \in q.\ (s,s') \in [C]\varepsilon$$

$$\frac{\vdash_F [p]\ C_1\ [er: q]}{\vdash_F [p]\ C_1; C_2\ [er: q]}$$

$$\frac{\vdash_F [p]\ C_1\ [ok: r] \qquad \vdash_F [r]\ C_2\ [\varepsilon: q]}{\vdash_F [p]\ C_1; C_2\ [\varepsilon: q]}$$

$$\frac{\vdash_F [p_1]\ C\ [\varepsilon: q_1] \qquad \vdash_F [p_2]\ C\ [\varepsilon: q_2]}{\vdash_F [p_1 \vee p_2]\ C\ [\varepsilon: q_1 \vee q_2]}$$

$$\frac{\vdash_F [p]\ C_i\ [\varepsilon: q] \qquad \textbf{some } i \in \{1, 2\}}{\vdash_F [p]\ C_1 + C_2\ [\varepsilon: q]}$$

$$\frac{}{\vdash_F [p]\ C^*\ [ok: p]}$$

$$\frac{\vdash_F [p]\ C^*; C\ [\varepsilon: q]}{\vdash_F [p]\ C^*\ [\varepsilon: q]}$$

FUX is **Under-Approximate**!

$$\vdash_F [p]\ C\ [\varepsilon: q] \quad \textit{iff} \quad \forall s \in p.\ \exists s' \in q.\ (s,s') \in [C]\varepsilon$$

$$\frac{\vdash_{BF} [p]\ C_1\ [er: q]}{\vdash_{BF} [p]\ C_1;\ C_2\ [er: q]}$$

$$\frac{\vdash_{BF} [p]\ C_1\ [ok: r] \qquad \vdash_{BF} [r]\ C_2\ [\varepsilon: q]}{\vdash_{BF} [p]\ C_1;\ C_2\ [\varepsilon: q]}$$

$$\frac{\vdash_{BF} [p_1]\ C\ [\varepsilon: q_1] \quad \vdash_{BF} [p_2]\ C\ [\varepsilon: q_2]}{\vdash_{BF} [p_1 \vee p_2]\ C\ [\varepsilon: q_1 \vee q_2]}$$

$$\frac{\vdash_{BF} [p]\ C_i\ [\varepsilon: q] \qquad \textbf{some}\ i \in \{1, 2\}}{\vdash_{BF} [p]\ C_1 + C_2\ [\varepsilon: q]}$$

$$\frac{}{\vdash_{BF} [p]\ C^*\ [ok: p]}$$

$$\frac{\vdash_{BF} [p]\ C^*;\ C\ [\varepsilon: q]}{\vdash_{BF} [p]\ C^*\ [\varepsilon: q]}$$

FUX is **Under-Approximate**!

$\vdash_F [p]\ C\ [\varepsilon: q]$ *iff* $\forall s \in p.\ \exists s' \in q.\ (s,s') \in [C]\varepsilon$

Q: What is the difference between FUX and BUX reasoning?

A: Rule of Consequence

$$\frac{}{\vdash_{BF} [p]\ C^*\ [ok: p]}$$

$$\frac{\vdash_{BF} [p]\ C^*; C\ [\varepsilon: q]}{\vdash_{BF} [p]\ C^*\ [\varepsilon: q]}$$

BUX vs. FUX

(ConsB)

$$\frac{p' \subseteq p \quad \vdash_B [p']\ C\ [\varepsilon : q'] \quad q' \supseteq q}{\vdash_B [p]\ C\ [\varepsilon : q]}$$

$\vdash_B [p]\ C\ [\varepsilon : q]$ *iff*

$\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

BUX vs. FUX

(ConsB)

$$\frac{p' \subseteq p \quad \vdash_B [p']\ C\ [\varepsilon : q'] \quad q' \supseteq q}{\vdash_B [p]\ C\ [\varepsilon : q]}$$

(ConsF)

$$\frac{p' \supseteq p \quad \vdash_F [p']\ C\ [\varepsilon : q'] \quad q' \subseteq q}{\vdash_F [p]\ C\ [\varepsilon : q]}$$

$\vdash_B [p]\ C\ [\varepsilon : q]$ *iff*

$\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

$\vdash_F [p]\ C\ [\varepsilon : q]$ *iff*

$\forall\, s \in p.\ \exists\, s' \in q.\ (s, s') \in [C]\varepsilon$

BUX vs. FUX

(ConsB)

$$\frac{p' \subseteq p \quad \vdash_B [p']\ C\ [\varepsilon: q'] \quad q' \supseteq q}{\vdash_B [p]\ C\ [\varepsilon: q]}$$

(ConsF)

$$\frac{p' \supseteq p \quad \vdash_F [p']\ C\ [\varepsilon: q'] \quad q' \subseteq q}{\vdash_F [p]\ C\ [\varepsilon: q]}$$

$\vdash_B [p]\ C\ [\varepsilon: q]$ *iff*

$\forall s \in q.\ \exists s' \in p.\ (s', s) \in [C]\varepsilon$

$\vdash_F [p]\ C\ [\varepsilon: q]$ *iff*

$\forall s \in p.\ \exists s' \in q.\ (s, s') \in [C]\varepsilon$

$$\frac{\vdash_B [p]\ C\ [\varepsilon: q_1 \vee q_2]}{\vdash_B [p]\ C\ [\varepsilon: q_1]}$$

Shrink the **post**

BUX vs. FUX

(ConsB)

$$\frac{p' \subseteq p \quad \vdash_B [p']\ C\ [\varepsilon: q'] \quad q' \supseteq q}{\vdash_B [p]\ C\ [\varepsilon: q]}$$

(ConsF)

$$\frac{p' \supseteq p \quad \vdash_F [p']\ C\ [\varepsilon: q'] \quad q' \subseteq q}{\vdash_F [p]\ C\ [\varepsilon: q]}$$

$\vdash_B [p]\ C\ [\varepsilon: q]$ *iff*

$\forall s \in q.\ \exists s' \in p.\ (s', s) \in [C]\varepsilon$

$\vdash_F [p]\ C\ [\varepsilon: q]$ *iff*

$\forall s \in p.\ \exists s' \in q.\ (s, s') \in [C]\varepsilon$

$$\frac{\vdash_B [p]\ C\ [\varepsilon: q_1 \vee q_2]}{\vdash_B [p]\ C\ [\varepsilon: q_1]}$$

Shrink the **post**

$$\frac{\vdash_F [p_1 \vee p_2]\ C\ [\varepsilon: q]}{\vdash_F [p_1]\ C\ [\varepsilon: q]}$$

Shrink the **pre**

BUX vs. FUX

(ConsB)

$p' \subseteq p \quad \vdash_B [p'] \; C \; [\varepsilon : q'] \quad q' \supseteq q$

(ConsF)

$p' \supseteq p \quad \vdash_F [p'] \; C \; [\varepsilon : q'] \quad q' \subseteq q$

Problem

Want to use existing **UX tools** (e.g. Pulse)

based on BUX

How to **practically reconcile BUX & FUX**?

$\vdash_B [p] \; C \; [\varepsilon : q_1]$

$\vdash_F [p_1] \; C \; [\varepsilon : q]$

Shrink the **post**

Shrink the **pre**

When are Disj and ConsB used in BUX?

$$\frac{\vdash_{BF} [p_1]\ C\ [\varepsilon: q_1] \quad \vdash_{BF} [p_2]\ C\ [\varepsilon: q_2]}{\vdash_{BF} [p_1 \lor p_2]\ C\ [\varepsilon: q_1 \lor q_2]}$$

- **Disj on paper**: to combine multiple triples
- **ConsB on paper**: to weaken pre or strengthen post

When are Disj and ConsB used in BUX?

$$\frac{\vdash_{BF} [p_1]\ C\ [\varepsilon: q_1] \quad \vdash_{BF} [p_2]\ C\ [\varepsilon: q_2]}{\vdash_{BF} [p_1 \lor p_2]\ C\ [\varepsilon: q_1 \lor q_2]}$$

- **Disj on paper**: to combine multiple triples
- **ConsB on paper**: to weaken pre or strengthen post
- **Disj in Pulse**: rarely used; pre-post correspondence tracked (distinct summaries)

When are Disj and ConsB used in BUX?

$$\frac{\vdash_{BF} [p_1]\ C\ [\varepsilon\!: q_1] \quad \vdash_{BF} [p_2]\ C\ [\varepsilon\!: q_2]}{\vdash_{BF} [p_1 \vee p_2]\ C\ [\varepsilon\!: q_1 \vee q_2]} \qquad \frac{\vdash_{B} [p]\ C\ [\varepsilon\!: q_1 \vee q_2]}{\vdash_{B} [p]\ C\ [\varepsilon\!: q_1]}$$

- **Disj on paper**: to combine multiple triples
- **ConsB on paper**: to weaken pre or strengthen post
- **Disj in Pulse**: rarely used; pre-post correspondence tracked (distinct summaries)
- **ConsB in Pulse**: mainly to drop disjuncts (i.e. forget summaries)

Indexed Disjuncts

$$P, Q \in \mathbb{N} \rightarrow \mathscr{P}(\text{States})$$

$$Q \equiv \bigvee_{i \in \text{dom}(Q)} q_i$$

Indexed Disjuncts

$$P, Q \in \mathbb{N} \rightarrow \mathscr{P}(\text{States}) \qquad Q \equiv \bigvee_{i \in dom(Q)} q_i$$

$$\vdash_{\dagger} [P]\ C\ [\varepsilon\text{: } Q] \quad \textit{iff} \quad dom(P) = dom(Q) \land$$
$$\forall\, i \in dom(P).\ \vdash_{\dagger} [P(i)]\ C\ [\varepsilon\text{: } Q(i)]$$

Unified BUX/FUX Framework

$$\frac{\vdash_{BF} [p_1]\ C\ [\varepsilon: q_1] \qquad \vdash_{BF} [p_2]\ C\ [\varepsilon: q_2]}{\vdash_{BF} [p_1 \vee p_2]\ C\ [\varepsilon: q_1 \vee q_2]} \rightsquigarrow \frac{\vdash_{BF} [P_1]\ C\ [\varepsilon: Q_1] \qquad \vdash_{BF} [P_2]\ C\ [\varepsilon: Q_2]}{\vdash_{BF} [P_1 \uplus P_2]\ C\ [\varepsilon: Q_1 \uplus Q_2]}$$

Unified BUX/FUX Framework

$$\frac{\vdash_{BF} [p_1]\ C\ [\varepsilon: q_1] \qquad \vdash_{BF} [p_2]\ C\ [\varepsilon: q_2]}{\vdash_{BF} [p_1 \vee p_2]\ C\ [\varepsilon: q_1 \vee q_2]} \rightsquigarrow \frac{\vdash_{BF} [P_1]\ C\ [\varepsilon: Q_1] \qquad \vdash_{BF} [P_2]\ C\ [\varepsilon: Q_2]}{\vdash_{BF} [P_1 \uplus P_2]\ C\ [\varepsilon: Q_1 \uplus Q_2]}$$

(ConsB)

$$\frac{p' \subseteq p \qquad \vdash_{B} [p']\ C\ [\varepsilon: q'] \qquad q' \supseteq q}{\vdash_{B} [p]\ C\ [\varepsilon: q]}$$

(ConsF)

$$\frac{p' \supseteq p \qquad \vdash_{F} [p']\ C\ [\varepsilon: q'] \qquad q' \subseteq q}{\vdash_{F} [p]\ C\ [\varepsilon: q]}$$

$$\rightsquigarrow \frac{\vdash_{BF} [P]\ C\ [\varepsilon: Q] \qquad I \subseteq \mathrm{dom}(P)}{\vdash_{BF} [P \downarrow I]\ C\ [\varepsilon: Q \downarrow I]}$$

Unified BUX/FUX Framework

Can use Pulse **as is**!

☞ **Extend** Pulse w. **divergence rules**

Relating BUX and FUX

Theorem 1.

$$\vdash_B [p]\ C\ [\varepsilon\colon q] \wedge \mathrm{minpre}(p, C, q) \;\Rightarrow\; \vdash_F [p]\ C\ [\varepsilon\colon q]$$

Relating BUX and FUX

Theorem 1.

$$\vdash_B [p]\ C\ [\varepsilon\!: q] \wedge \mathrm{minpre}(p, C, q) \;\Rightarrow\; \vdash_F [p]\ C\ [\varepsilon\!: q]$$

where $\mathrm{minpre}(p, C, q)$ ***iff*** $\forall p'.\ \vdash_B [p']\ C\ [\varepsilon\!: q] \Rightarrow p' \not\subset p$

Relating BUX and FUX

Theorem 1.

$\vdash_B [p]\ C\ [\varepsilon: q] \land \mathrm{minpre}(p, C, q) \Rightarrow \vdash_F [p]\ C\ [\varepsilon: q]$

where $\mathrm{minpre}(p, C, q)$ ***iff*** $\forall p'. \vdash_B [p']\ C\ [\varepsilon: q] \Rightarrow p' \not\subset p$

Theorem 2.

$\vdash_F [p]\ C\ [\varepsilon: q] \land \mathrm{minpost}(p, C, q) \Rightarrow \vdash_B [p]\ C\ [\varepsilon: q]$

where $\mathrm{minpost}(p, C, q)$ ***iff*** $\forall q'. \vdash_F [p]\ C\ [\varepsilon: q'] \Rightarrow q' \not\subset q$

The soundness of bugs is what matters!

The goal is to find bugs!

"Most program analysis & verification research seems confused about the ultimate goal of software defect detection. The main practical usefulness of such techniques is the ability to find bugs, not to report that no bugs have been found."

Patrice Godefroid, 2005

The soundness of bugs is what matters!

The goal is to find bugs!

"Most program analysis & verification research seems confused about the ultimate goal of software defect detection. The main practical usefulness of such techniques is the ability to find bugs, not to report that no bugs have been found."

Patrice Godefroid, 2005

Thank You for Listening!

azalea@imperial.ac.uk SoundAndComplete.org @azalearaad