Accumulation Analysis

Martin Kellogg[a], Narges Shadab[b], Manu Sridharan[b], Michael D. Ernst[a]

[a]University of Washington [b]University of California, Riverside

Problem: sound typestate analysis is expensive

Problem: sound typestate analysis is expensive

- **Accumulation typestate automata** are exactly those that can be checked **without aliasing information**, the **traditional bottleneck** for a typestate analysis

Problem: sound typestate analysis is expensive

- **Accumulation typestate automata** are exactly those that can be checked **without aliasing information**, the **traditional bottleneck** for a typestate analysis
- Accumulation typestate automata include **important problems** like resource leaks, security vulnerabilities, and initialization

Problem: sound typestate analysis is expensive

- **Accumulation typestate automata** are exactly those that can be checked **without aliasing information**, the **traditional bottleneck** for a typestate analysis
- Accumulation typestate automata include **important problems** like resource leaks, security vulnerabilities, and initialization
- For accumulation typestate problems, an accumulation analysis is **sound, precise, and fast**

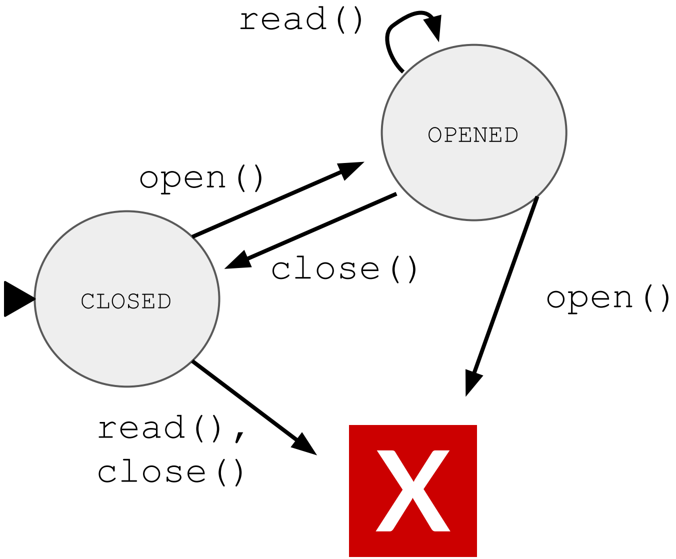
Talk outline

- Background on typestate
- Accumulation analysis
  - definitions & examples
  - proofs
- Literature survey
- Implications for practicality

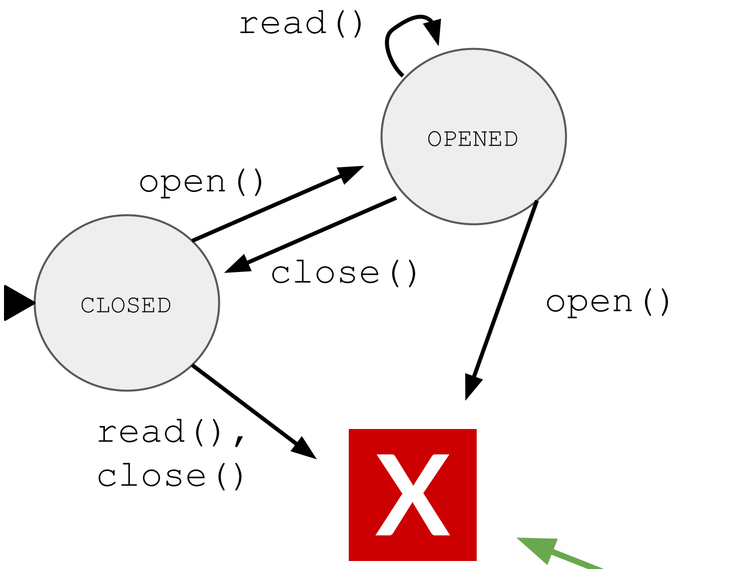
Typestate analysis

- Classic static program analysis technique (Strom & Yemeni, 1986)
- Extensive literature: **over 18,000** hits on Google Scholar

Typestate specification via FSM

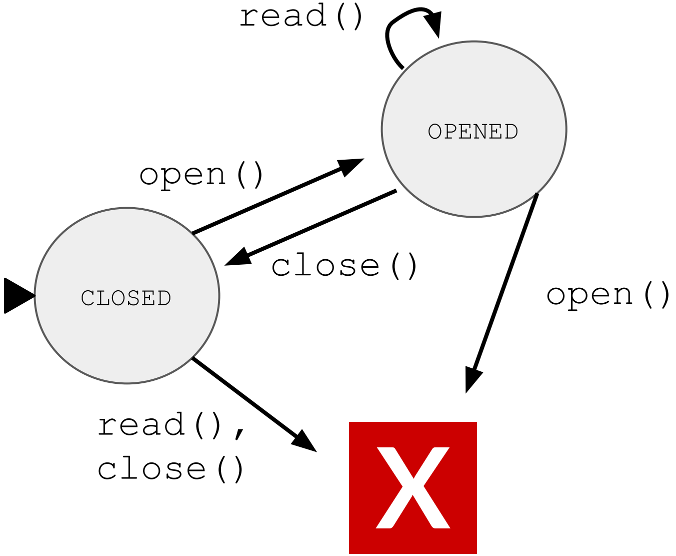


Typestate specification via FSM



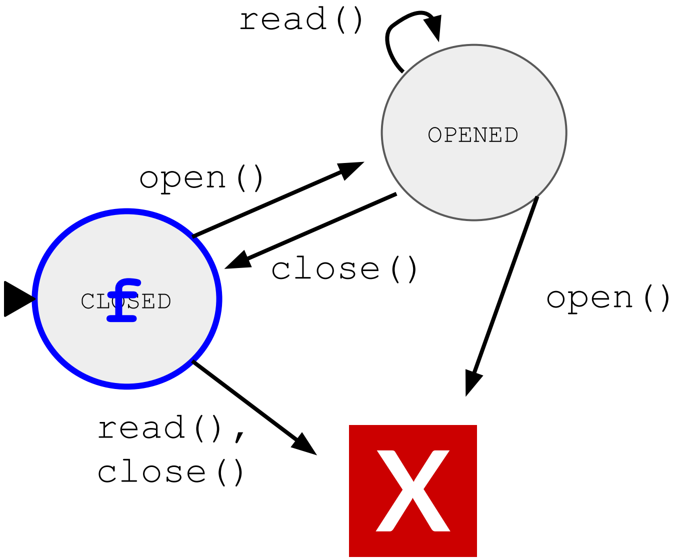
Our goal: ***prove*** that no File ever enters this state

Typestate specification via FSM



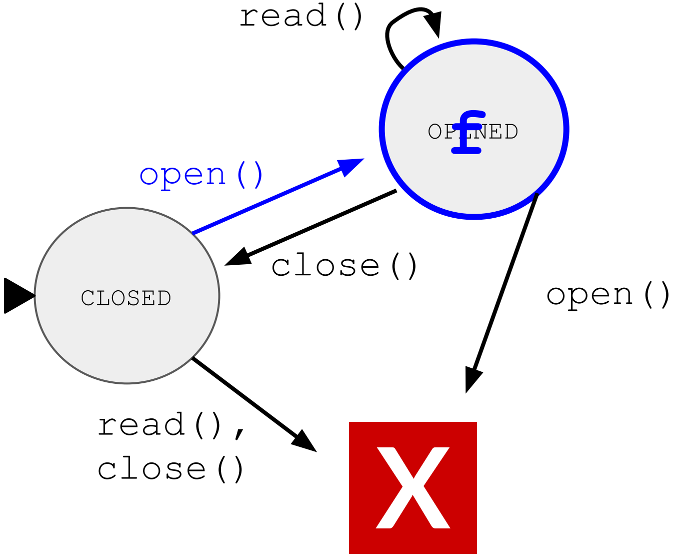

```
File f = …;
f.open();
f.close();
f.read();
```

Typestate specification via FSM



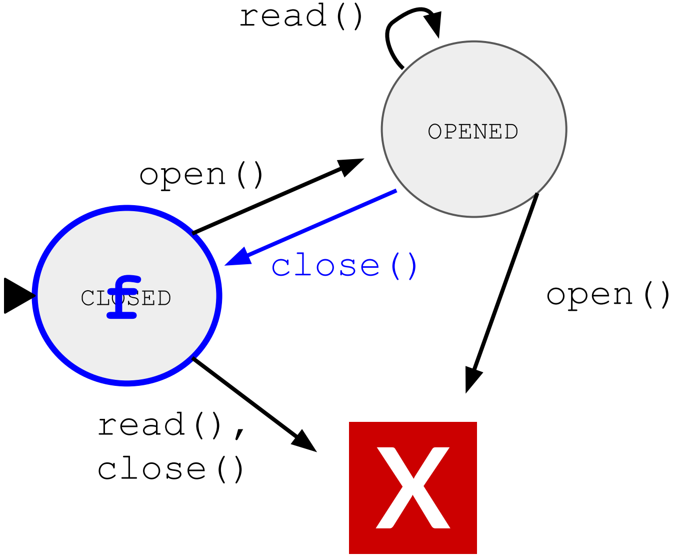

```
File f = …;
f.open();
f.close();
f.read();
```

Typestate specification via FSM



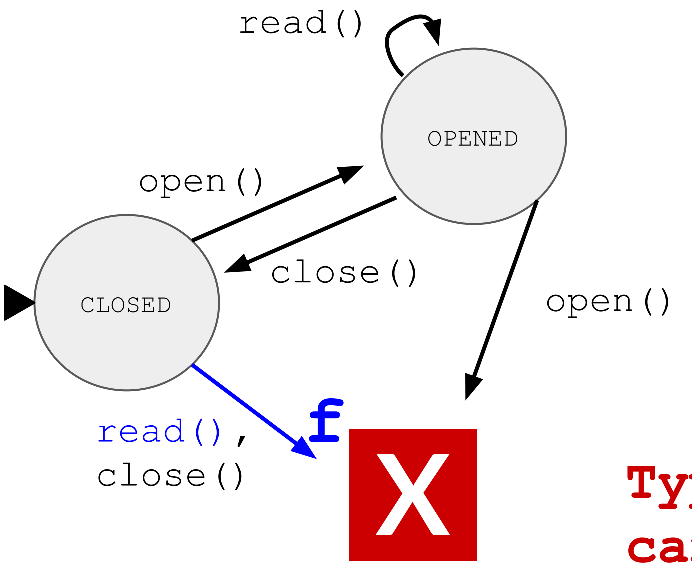

```
File f = …;
f.open();
f.close();
f.read();
```

Typestate specification via FSM




```
File f = …;
f.open();
f.close();
f.read();
```

Typestate specification via FSM




```
File f = …;
f.open();
f.close();
f.read();
```

Typestate error: f cannot read() in state CLOSED

Sound typestate requires aliasing information

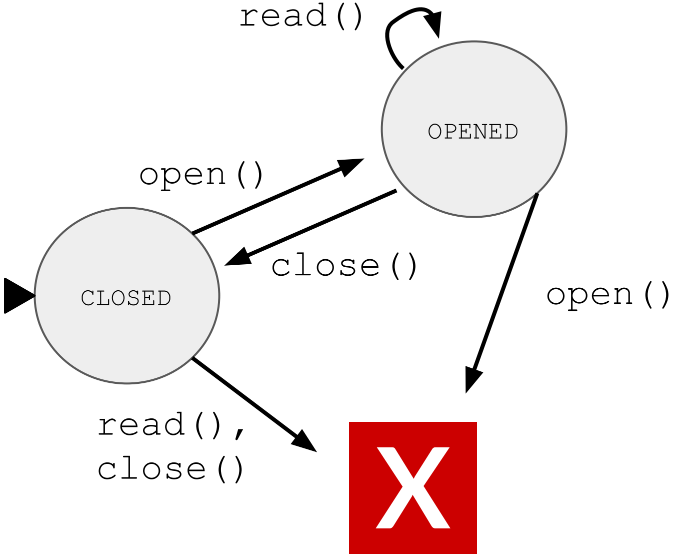
- A **sound** typestate analysis must **track all aliases** to keep FSMs in sync

Sound typestate requires aliasing information

- A **sound** typestate analysis must **track all aliases** to keep FSMs in sync

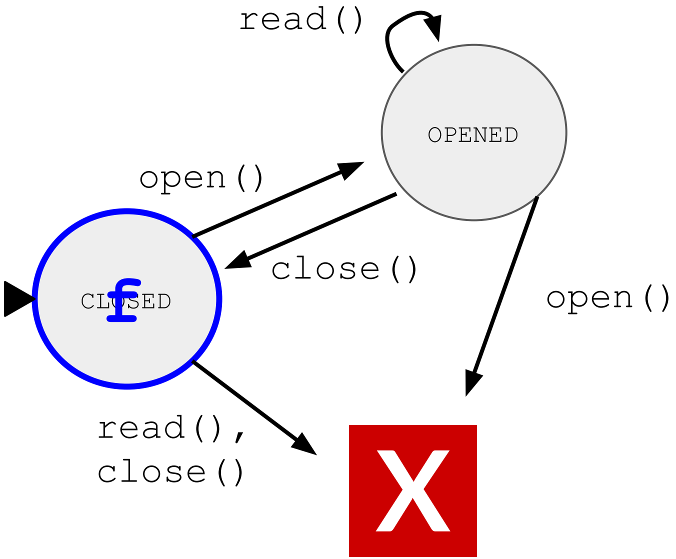
Soundness is **important**:
- enables verification vs. bug finding
- mission-critical domains

Why is typestate expensive?



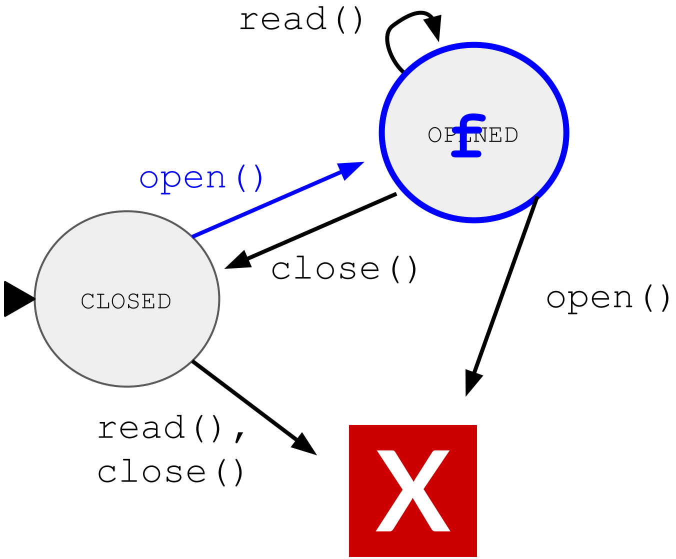

```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

Why is typestate expensive?



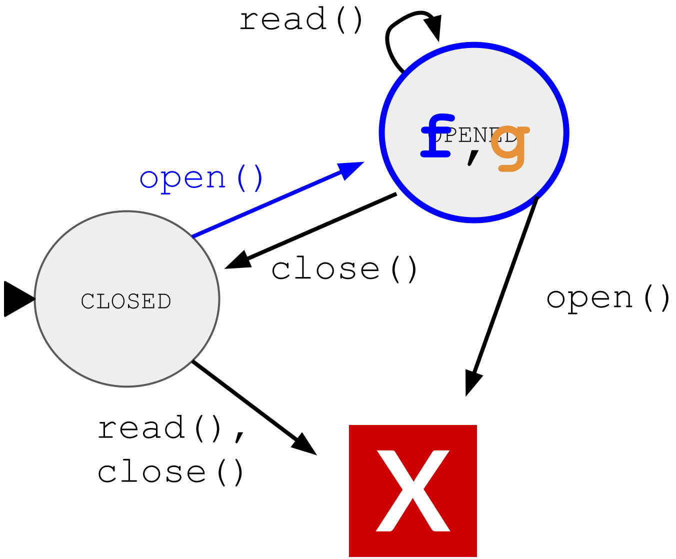

```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

Why is typestate expensive?



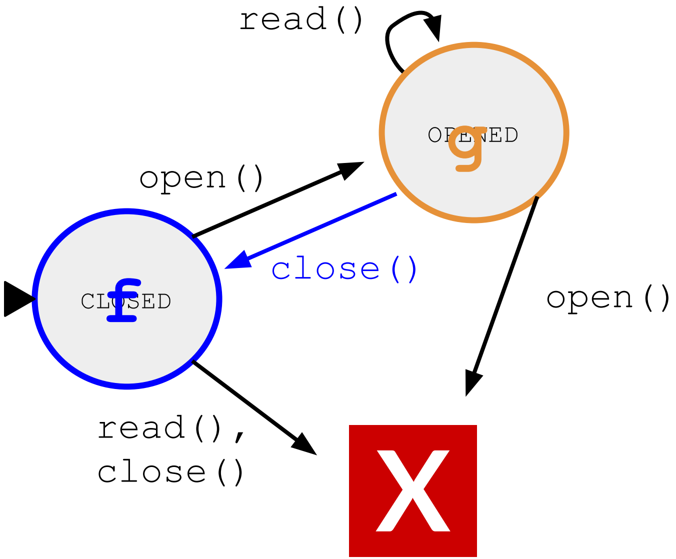

```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

Why is typestate expensive?



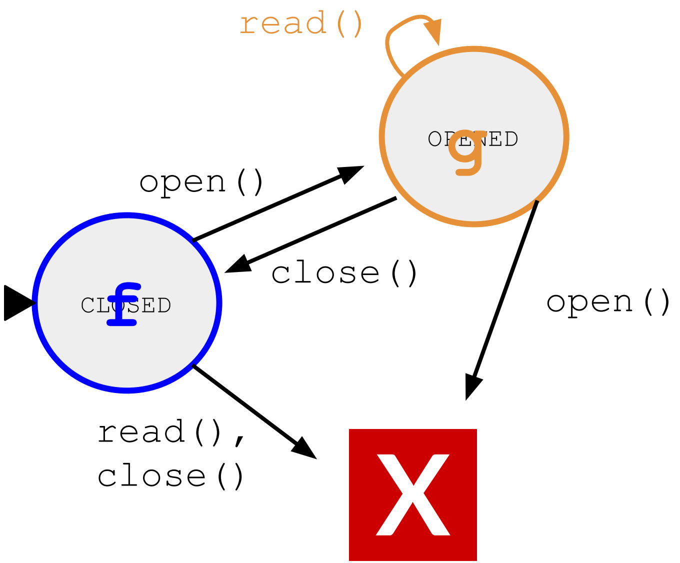

```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

Why is typestate expensive?



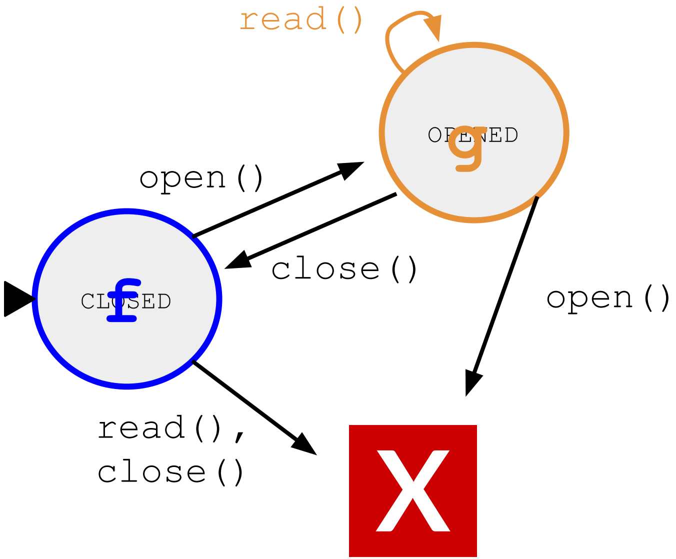

```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

Why is typestate expensive?




```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

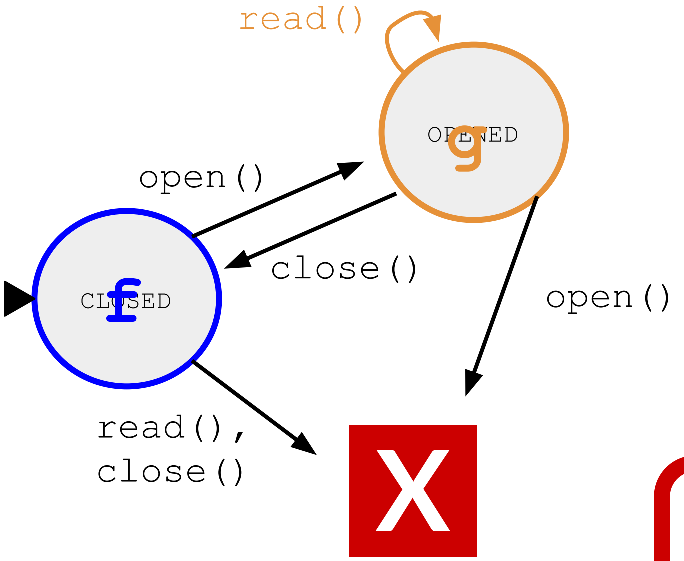
Why is typestate expensive? Aliasing.




```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

“false negative”

Why is typestate expensive? Aliasing.




```
File f = …;
f.open();
File g = f;
f.close();
g.read();
```

"false negative" = <u>unsound</u>!

Sound typestate requires aliasing information

- A **sound** typestate analysis must **track all aliases** to keep FSMs in sync

Sound typestate requires aliasing information

- A **sound** typestate analysis must **track all aliases** to keep FSMs in sync
- Three prior approaches:

Sound typestate requires aliasing information

- A **sound** typestate analysis must **track all aliases** to keep FSMs in sync
- Three prior approaches:
  1. **whole-program** may-alias analysis (expensive)

     → Tan et al. 2021 report hours for real programs

Sound typestate requires aliasing information

- A **sound** typestate analysis must **track all aliases** to keep FSMs in sync
- Three prior approaches:
    1. **whole-program** may-alias analysis (expensive)
    2. **restrict aliasing** (e.g., via ownership types)

        → e.g., Bierhoff et al. 2009, Clark et al. 2013, Rust

Sound typestate requires aliasing information

- A **sound** typestate analysis must **track all aliases** to keep FSMs in sync
- Three prior approaches:
  1. **whole-program** may-alias analysis (expensive)
  2. **restrict aliasing** (e.g., via ownership types)
  3. **ignore aliasing** and be unsound (due to cost)

     → allows industry deployment, e.g., Emmi et al. 2021

Sound typestate requires aliasing information

- A **sound** typestate analysis must **track all aliases** to keep FSMs in sync
- Three prior approaches:
  1. **whole-program** may-alias analysis (expensive)
  2. **restrict aliasing** (e.g., via ownership types)
  3. **ignore aliasing** and be unsound (due to cost)

Key question: does typestate analysis *always* need aliasing information?

Insight: aliasing information is only required for some typestate automata

Insight: aliasing information is only required for some typestate automata

Which ones?

Insight: aliasing information is only required for some typestate automata

Which ones?

Key intuition: once an operation ***becomes legal***, it ***stays legal***

Accumulation typestates

accumulation typestate automaton:

for any **error-inducing sequence** $S = t_1, \ldots, t_i$,

all **subsequences** of S that end in t_i

are also **error-inducing**

Accumulation typestates

accumulation typestate automaton:

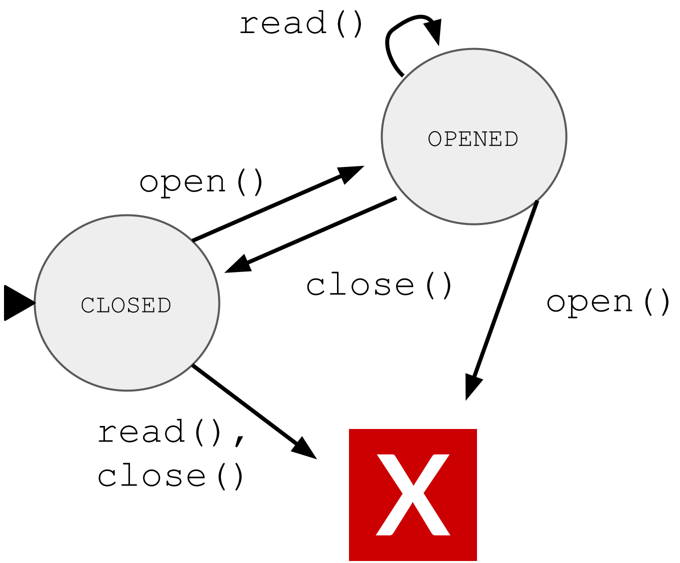
for any **error-inducing sequence** $S = t_1, \ldots, t_i$,

all **subsequences** of *S* that end in t_i

are also **error-inducing**

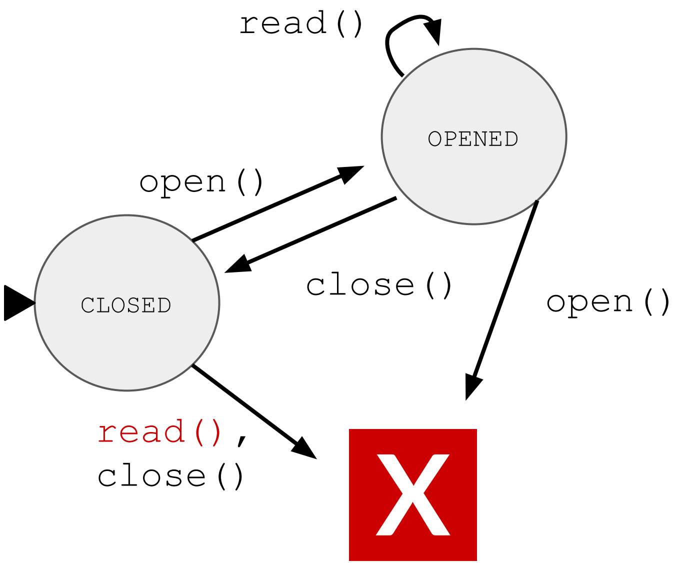
Key theorem: Accumulation typestates are **exactly** those that can be checked soundly **without aliasing information**

Is it an accumulation typestate automaton?



for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
all **subsequences** of S that end in t_i
are also **error-inducing**

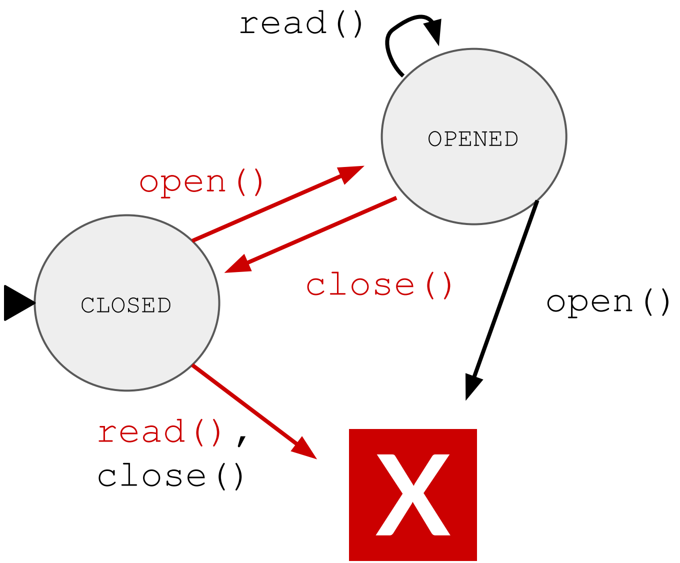
Is it an accumulation typestate automaton?



for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
all **subsequences** of S that end in t_i
are also **error-inducing**

S = `read()`

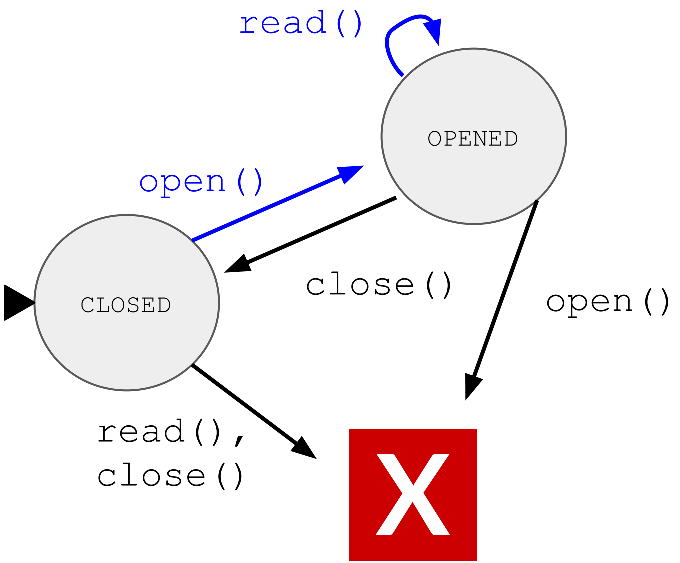
Is it an accumulation typestate automaton?



for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
all **subsequences** of S that end in t_i
are also **error-inducing**

S = `open(), close(), read().`

Is it an accumulation typestate automaton?

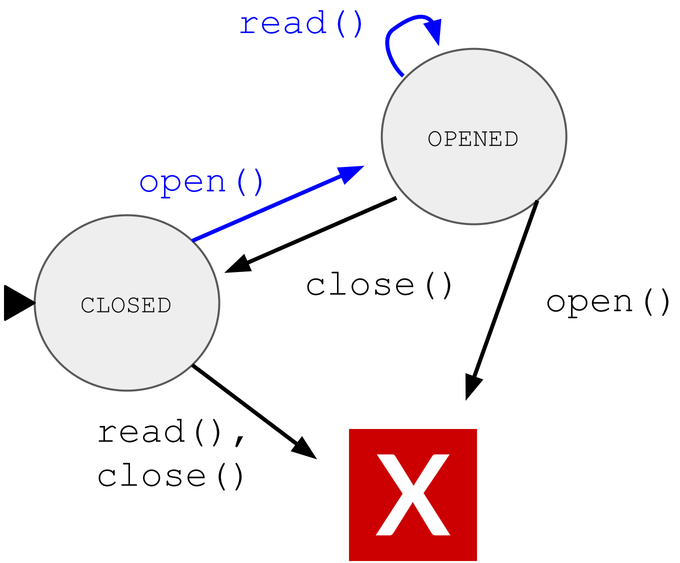


for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
all **subsequences** of S that end in t_i
are also **error-inducing**

S = `open(), close(), read().`

S′ = `open(),` ~~`close()`~~, `read()`
is not error-inducing!

Is it an accumulation typestate automaton?



for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
all **subsequences** of *S* that end in t_i
are also **error-inducing**

S = `open(), close(), read().`

S′ = `open(), ~~close()~~, read()`
is not error-inducing!
⇒ **not accumulation**

Is it an accumulation typestate automaton?

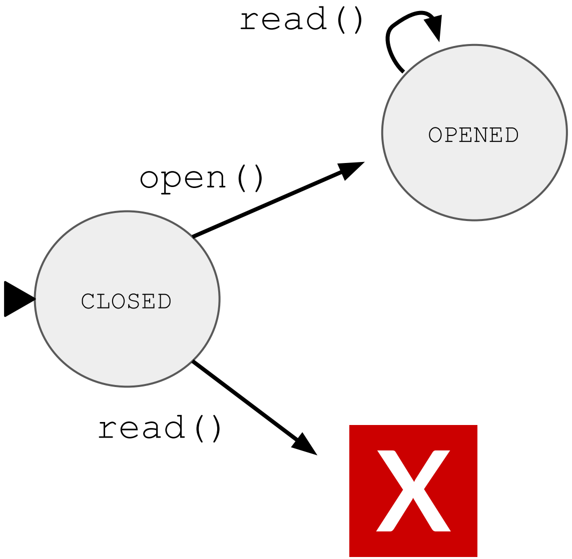
"only call `read()`
after calling `open()`"

for any **error-inducing sequence** *S* = $t_1, \ldots, t_i$,
all **subsequences** of *S* that end in t_i
are also **error-inducing**

Is it an accumulation typestate automaton?

"only call `read()`
after calling `open()`"

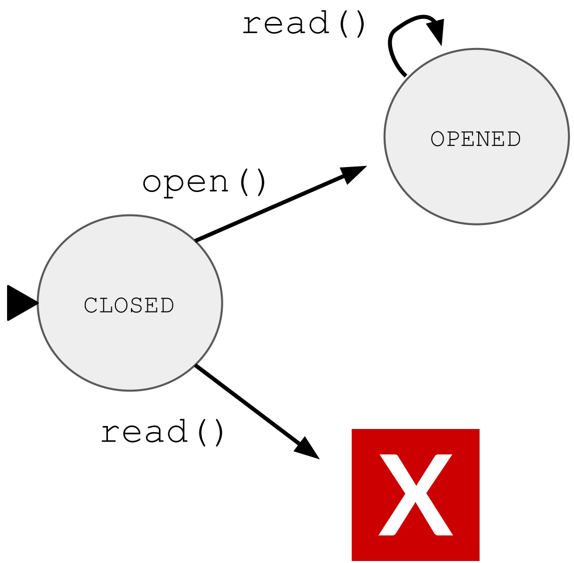
for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
all **subsequences** of S that end in t_i
are also **error-inducing**



Is it an accumulation typestate automaton?

"only call `read()`
after calling `open()`"

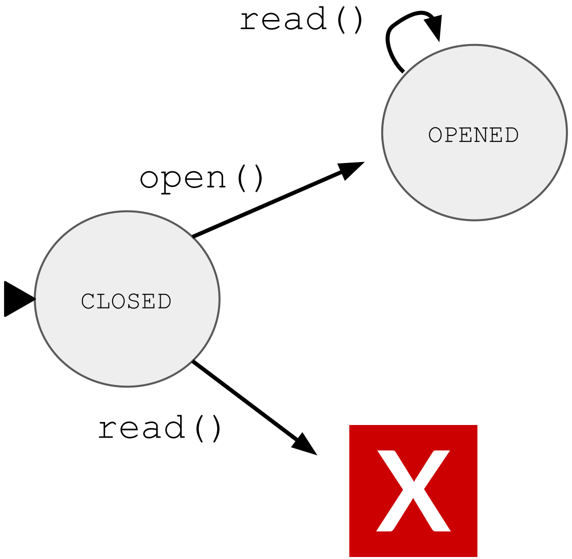
for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
all **subsequences** of S that end in t_i
are also **error-inducing**



S = `read()`

Is it an accumulation typestate automaton?

"only call `read()` after calling `open()`"



for any **error-inducing sequence** $S = t_1, \ldots, t_i$, all **subsequences** of S that end in t_i are also **error-inducing**

S = `read()`

⇒ **YES accumulation!**

Aside: how hard is it to decide if a typestate automaton is accumulation?

Aside: how hard is it to decide if a typestate automaton is accumulation?

- As easy as checking DFA equivalence
  - Result due to **Higman's Theorem** (1952)

Aside: how hard is it to decide if a typestate automaton is accumulation?

- As easy as checking DFA equivalence
  - Result due to **Higman's Theorem** (1952)

> "The subsequence language of *any language whatsoever* over a finite alphabet is regular."

Accumulation typestates

accumulation typestate automaton:

for any **error-inducing sequence** $S = t_1, \ldots, t_i$,

all **subsequences** of S that end in t_i

are also **error-inducing**

Key theorem: Accumulation typestates are **exactly** those that can be checked soundly **without aliasing information**

Accumulation typestates

accumulation typestate automaton:

for any **error-inducing sequence** $S = t_1, \ldots, t_i$,

all **subsequences** of S that end in t_i

are also **error-inducing**

Key theorem: Accumulation typestates are **exactly** those that can be checked soundly **without aliasing information**

Proof

Key theorem: Accumulation typestates are **exactly** those that can be checked soundly **without aliasing information**

Proof

Key theorem: Accumulation typestates are **exactly** those that can be checked soundly **without aliasing information**

1. ⇒ ("**all** accumulation typestates can be checked soundly without aliasing information")
2. ⇐ ("**only** accumulation typestates can be checked soundly without aliasing information")

Accumulation typestate ⇒ soundly checkable without aliasing information

Accumulation typestate ⇒ soundly checkable without aliasing information

1. without aliasing information, analysis **observes a subsequence** of actual transitions

Accumulation typestate ⇒ soundly checkable without aliasing information

1. without aliasing information, analysis **observes a subsequence** of actual transitions
2. if analysis observes a transition that leads to an error at run time, the final transition **must be error-inducing**

Accumulation typestate ⇒ soundly checkable without aliasing information

1. without aliasing information, analysis **observes a subsequence** of actual transitions
2. if analysis observes a transition that leads to an error at run time, the final transition **must be error-inducing**

> for any **error-inducing sequence** $S = t_1, \ldots, t_i$,
> all **subsequences** of S that end in t_i
> are also **error-inducing**

Soundly checkable without aliasing information ⇒ accumulation typestate

Soundly checkable without aliasing information ⇒ accumulation typestate

1. **suppose** we have a non-accumulation typestate that can be checked without aliasing information

Soundly checkable without aliasing information ⇒ accumulation typestate

1. **suppose** we have a non-accumulation typestate that can be checked without aliasing information
2. this automaton has an **error-inducing sequence S** with a **non-error-inducing subsequence S'**

Soundly checkable without aliasing information ⇒ accumulation typestate

1. **suppose** we have a non-accumulation typestate that can be checked without aliasing information
2. this automaton has an **error-inducing sequence S** with a **non-error-inducing subsequence S'**

for any **error-inducing sequence** $S = t_1, \ldots, t_i$, all **subsequences** of S that end in t_i are also **error-inducing**

Soundly checkable without aliasing information ⇒ accumulation typestate

1. **suppose** we have a non-accumulation typestate that can be checked without aliasing information
2. this automaton has an **error-inducing sequence S** with a **non-error-inducing subsequence S'**

for any **err** [redacted] **inducing sequen** [redacted] $= t_1, \ldots, t_i$,
all **subsequenc** [redacted] in t_i
are also **er** [redacted] **-inducing**

Soundly checkable without aliasing information ⇒ accumulation typestate

1. **suppose** we have a non-accumulation typestate that can be checked without aliasing information
2. this automaton has an **error-inducing sequence S** with a **non-error-inducing subsequence S'**
3. construct a program with two aliased variables: do **S - S' on the first**, and **S' on the second**

Soundly checkable without aliasing information ⇒ accumulation typestate

1 $x_1 = x_2$

2

3

Soundly checkable without aliasing information ⇒ accumulation typestate

1 $x_1 = x_2$ ← both point to a single value ***v***

2

3

Soundly checkable without aliasing information ⇒ accumulation typestate

1 $x_1 = x_2$ ← both point to a single value ***v***

$\forall$ t $\in$ **S**:

2 if t $\in$ **S - S'**: x_1.t()

else if t $\in$ **S'**: x_2.t()

3

Soundly checkable without aliasing information ⇒ accumulation typestate

1 $x_1 = x_2$ ← both point to a single value ***v***

$\forall\, t \in$ **S**:

2 if $t \in$ **S - S'**: $x_1.t()$

else if $t \in$ **S'**: $x_2.t()$

3

contradiction: ***v*** must be in an error state (**S** is **error-inducing**), **but** analysis cannot warn about x_2 (**S'** is **non-error-inducing**)

Soundly checkable without aliasing information ⇒ accumulation typestate

1 $x_1 = x_2$ ← both point to a single value ***v***

$\forall\, t \in$ **S**:

2 if $t \in$ **S - S'**: $x_1.t()$

else if $t \in$ **S'**: $x_2.t()$

the "sound" analysis misses the real error!

3

contradiction: ***v*** must be in an error state (**S** is **error-inducing**), **but** analysis cannot warn about x_2 (**S'** is **non-error-inducing**)

Proof

Key theorem: Accumulation typestates are **exactly** those that can be checked soundly **without aliasing information**

1. ⇒ ("**all** accumulation typestates can be checked soundly without aliasing information")
2. ⇐ ("**only** accumulation typestates can be checked soundly without aliasing information")

How common is accumulation?

How common is accumulation?

- Literature survey of 188 typestate papers since 1999

How common is accumulation?

- Literature survey of 188 typestate papers since 1999
    - 85 with no typestate automata

How common is accumulation?

- Literature survey of 188 typestate papers since 1999
  - ~~85 with no typestate automata~~

How common is accumulation?

- Literature survey of 188 typestate papers since 1999
    - ~~85 with no typestate automata~~
    - 101 papers with < 20 examples

How common is accumulation?

- Literature survey of 188 typestate papers since 1999
  - ~~85 with no typestate automata~~
  - 101 papers with < 20 examples
  - 2 papers with categories of automata:
    - Dwyer et al. (ICSE 1999)
    - Beckman et al. (ECOOP 2011)

How common is accumulation?

- Literature survey of 188 typestate papers since 1999
  - ~~85 with no typestate automata~~
  - 101 papers with < 20 examples
  - 2 papers with categories of automata:
    - Dwyer et al. (ICSE 1999)
    - Beckman et al. (ECOOP 2011)

67 / 302

How common is accumulation?

- Literature survey of 188 typestate papers since 1999
  - ~~85 with no typestate automata~~
  - 101 papers with < 20 examples
  - 2 papers with categories of automata:
    - Dwyer et al. (ICSE 1999)
    - Beckman et al. (ECOOP 2011)

67 / 302

306 / 511

How common is accumulation?

- Literature survey of 188 typestate papers since 1999
  - ~~85 with no typestate automata~~
  - 101 papers with < 20 examples
  - 2 papers with categories of automata:
    - Dwyer et al. (ICSE 1999)
    - Beckman et al. (ECOOP 2011)

67 / 302

306 / 511

182 / 542

How common is accumulation: takeaways

- 555 / 1355 (**41%**) of typestate automata are accumulation

How common is accumulation: takeaways

- 555 / 1355 (**41%**) of typestate automata are accumulation
- Higher proportion of accumulation TSA in large collections: **more common in practice?**

How common is accumulation: takeaways

- 555 / 1355 (**41%**) of typestate automata are accumulation
- Higher proportion of accumulation TSA in large collections: **more common in practice?**
- Our artifact includes all the TSAs we saw

https://doi.org/10.5281/zenodo.5771196

Practicality of accumulation

Practicality of accumulation

Source LoC	~9.1M
True positives	16
False positives	3

Kellogg, Ran, Sridharan, Schaef, Ernst. *Verifying Object Construction*. ICSE 2020.

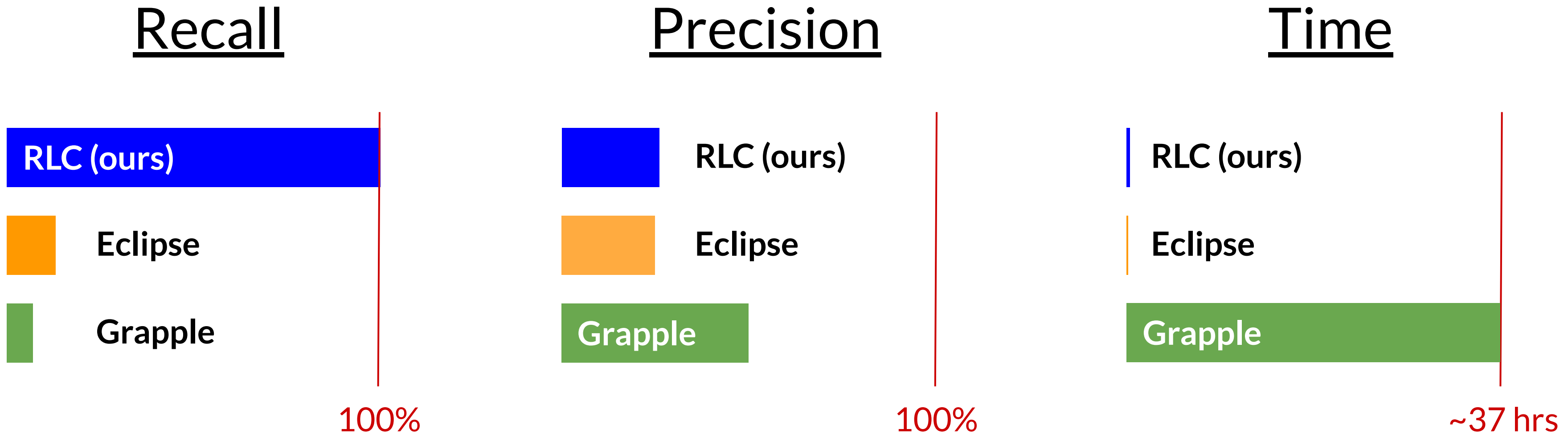
Practicality of accumulation

Source LoC	~9.1M
True positives	16
False positives	3

100% recall,
82% precision

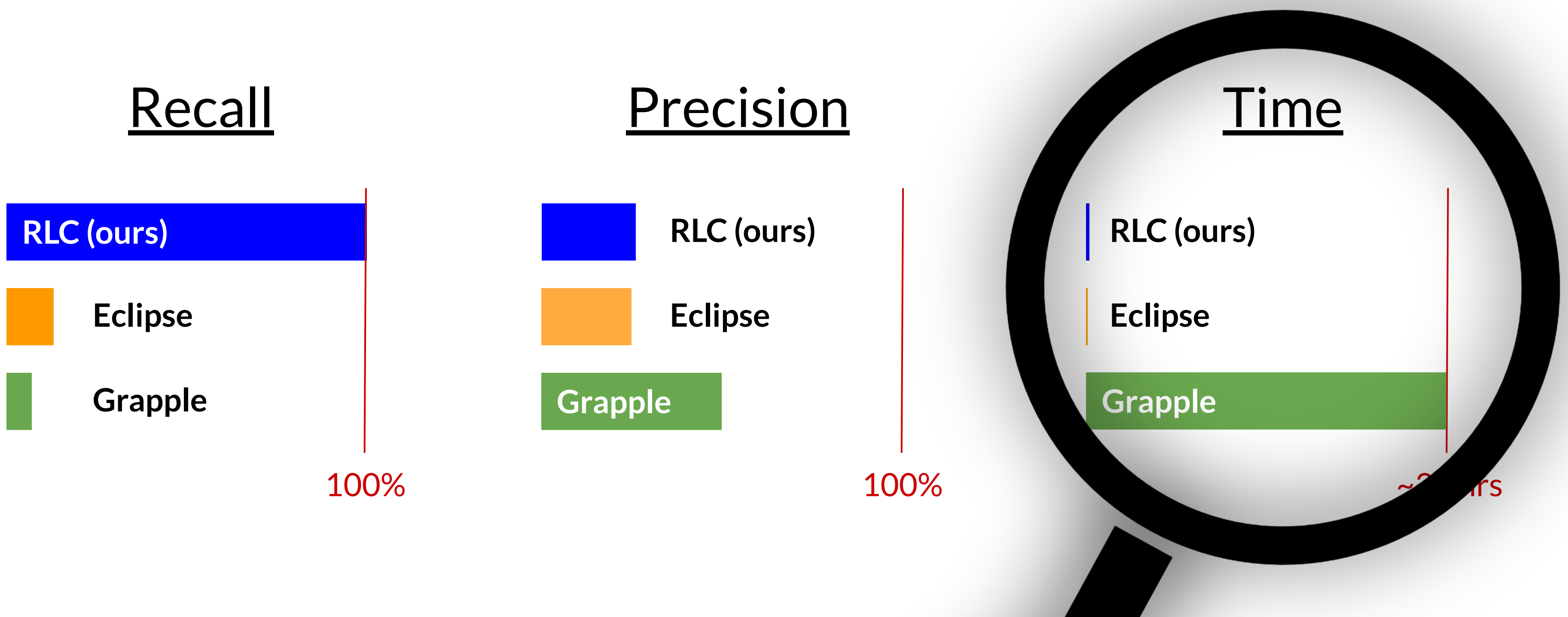
Kellogg, Ran, Sridharan, Schaef, Ernst. *Verifying Object Construction*. ICSE 2020.

Practicality of accumulation



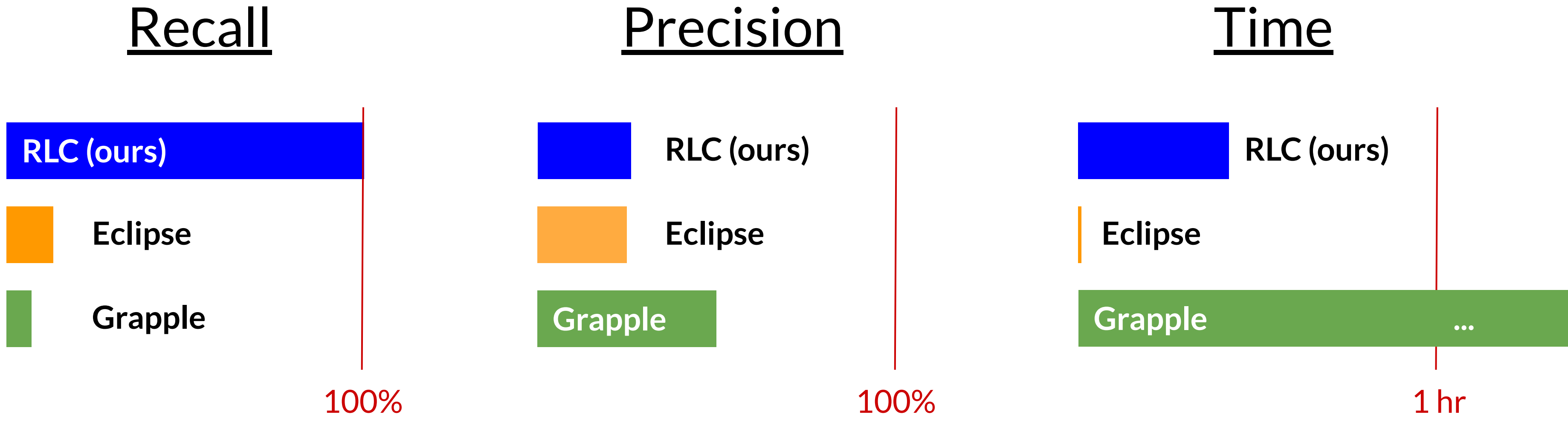
Kellogg, Shadab, Sridharan, Ernst. *Lightweight and Modular Resource Leak Verification*. ESEC/FSE 2021.

Practicality of accumulation



Kellogg, Shadab, Sridharan, Ernst. *Lightweight and Modular Resource Leak Verification.* ESEC/FSE 2021.

Practicality of accumulation



<u>Kellogg</u>, Shadab, Sridharan, Ernst. *Lightweight and Modular Resource Leak Verification*. ESEC/FSE 2021.

Practicality of accumulation

- Important lessons:

Practicality of accumulation

- Important lessons:
    - when accumulation is **applicable**, it produces analyses that are **sound, precise, and fast**

Practicality of accumulation

- Important lessons:
  - when accumulation is **applicable**, it produces analyses that are **sound, precise, and fast**
  - **cheap, local alias reasoning** is always useful for **precision**

Practicality of accumulation

- Important lessons:
  - when accumulation is **applicable**, it produces analyses that are **sound, precise, and fast**
  - **cheap, local alias reasoning** is always useful for **precision**
  - sound with no aliasing information ⇒ sound with limited aliasing information

Contributions

- Identification of the **accumulation typestate automata**, a new, important subset of typestates
- Proof that accumulation typestates are **exactly** those checkable **without aliasing information**
- **41%** of typestate automata are accumulation
- Practical accumulation analyses are **sound**, **precise**, and **fast**

Contributions

- Identification of the **accumulation typestate automata**, a new, important subset of typestates
- Proof that accumulation typestates are **exactly** those checkable **without aliasing information**
- **41%** of typestate automata are accumulation
- Practical accumulation analyses are **sound**, **precise**, and **fast**