# RELIA: Accelerating the Analysis of Cloud Access Control Policies

Dan Wang[1], Peng Zhang[1], Zhenrong Gu[1], Weibo Lin[2], Shibiao Jiang[2], Zhu He[2], Xu Du[2], Longfei Chen[2], Jun Li[2], and Xiaohong Guan[1]

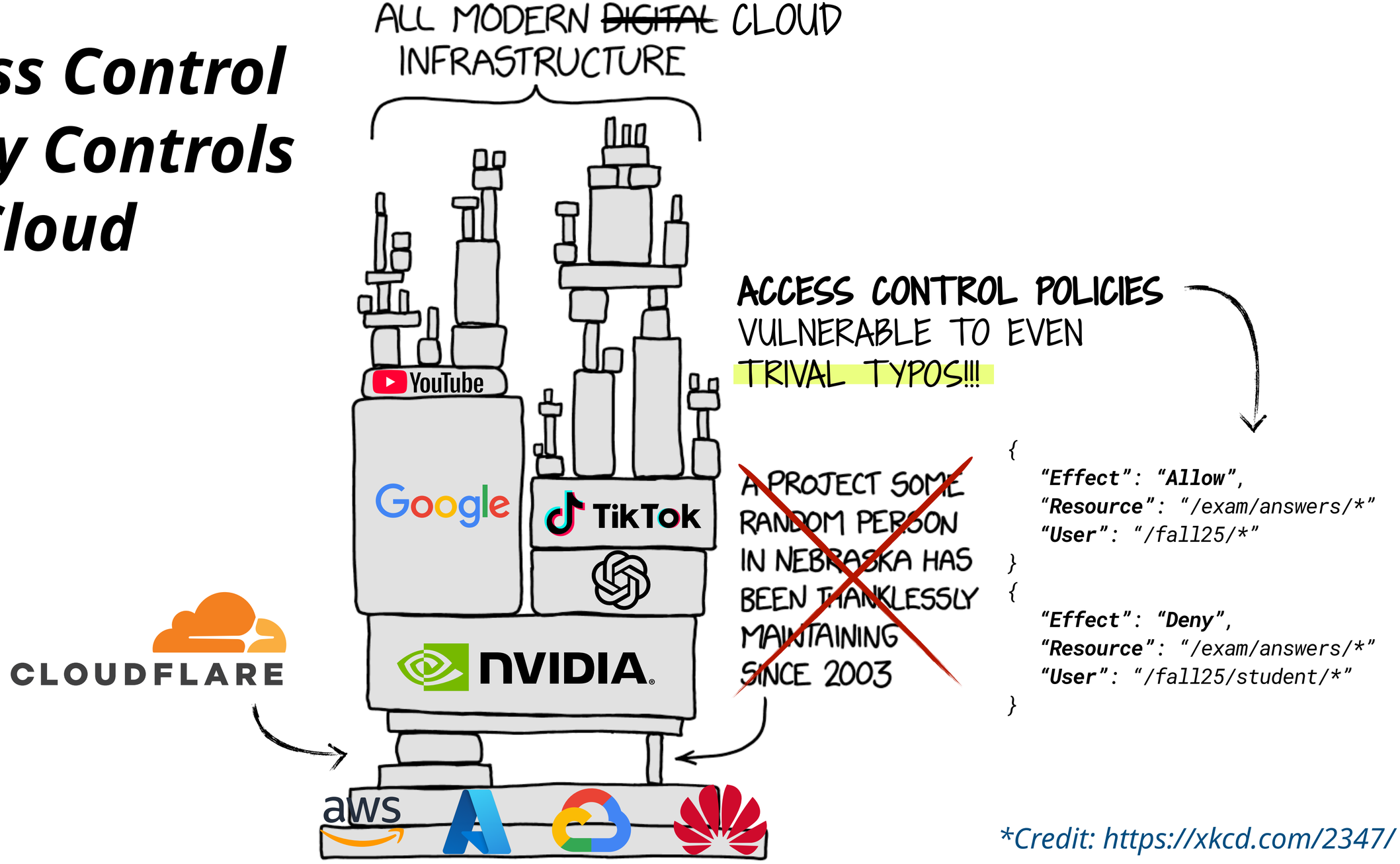
[1] 西安交通大學 XI'AN JIAOTONG UNIVERSITY

[2] HUAWEI

ASE 2025 *the* 40th IEEE/ACM International Conference on **Automated Software Engineering (ASE 2025)**

*Seoul, Korea*
*November 16 - 20, 2025*

# *Access Control Policy Controls the Cloud*




*Credit: https://xkcd.com/2347/

# *Cloud Access Control Policy Analyzers!*

*Analyzer: Zelkova [FMCAD '18]*

**Policies**

```
{
  "Effect": "Allow",
  "Resource": "/exam/answers/*"
  "User": "/fall25/*"
}
```

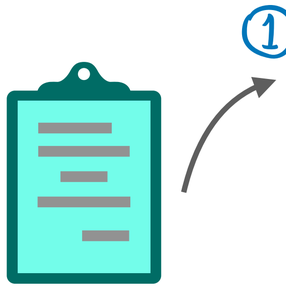
*"**Allow** the access of **exam answers** from **all members** of **fall 2025** semester;*

```
{
  "Effect": "Deny",
  "Resource": "/exam/answers/*"
  "User": "/fall25/student/*"
}
```

*but **do not allow** the access from those who are **students in fall 2025**"*

# *Cloud Access Control Policy Analyzers!*

*Analyzer: Zelkova [FMCAD '18]*



**Policies**

①

```
C = (and
     (str.in_re user "/fall25/*")
     (not (str.in_re user "/fall25/student/*")))
```

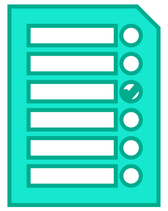
# Cloud Access Control Policy Analyzers!

Analyzer: Zelkova [FMCAD '18]

Policies

①

```
C = (and
     (str.in_re user "/fall25/*")
     (not (str.in_re user "/fall25/student/*")))
```



Intents

②

```
(assert
     (and C (= user "/fall25/admin/Alice")))
```

```
(assert (not
     (and C (= user "/fall25/student/Eve"))))
```

Findings: AccessSummary [CAV '20]

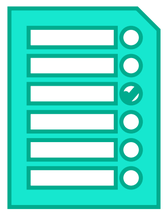
# Cloud Access Control Policy Analyzers!

Analyzer: Zelkova [FMCAD '18]

Policies

①

```
C = (and
     (str.in_re user "/fall25/*")
     (not (str.in_re user "/fall25/student/*")))
```



Intents

②

```
(assert
     (and C (= user "/fall25/admin/Alice")))

(assert (not
     (and C (= user "/fall25/student/Eve"))))
```

③

SMT Solver

Findings: AccessSummary [CAV '20]

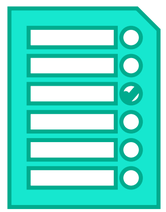
# Cloud Access Control Policy Analyzers!

Analyzer: Zelkova [FMCAD '18]

Policies

①

```
C = (and
     (str.in_re user "/fall25/*")
     (not (str.in_re user "/fall25/student/*")))
```



Intents

②

```
(assert
     (and C (= user "/fall25/admin/Alice")))

(assert (not
     (and C (= user "/fall25/student/Eve"))))
```

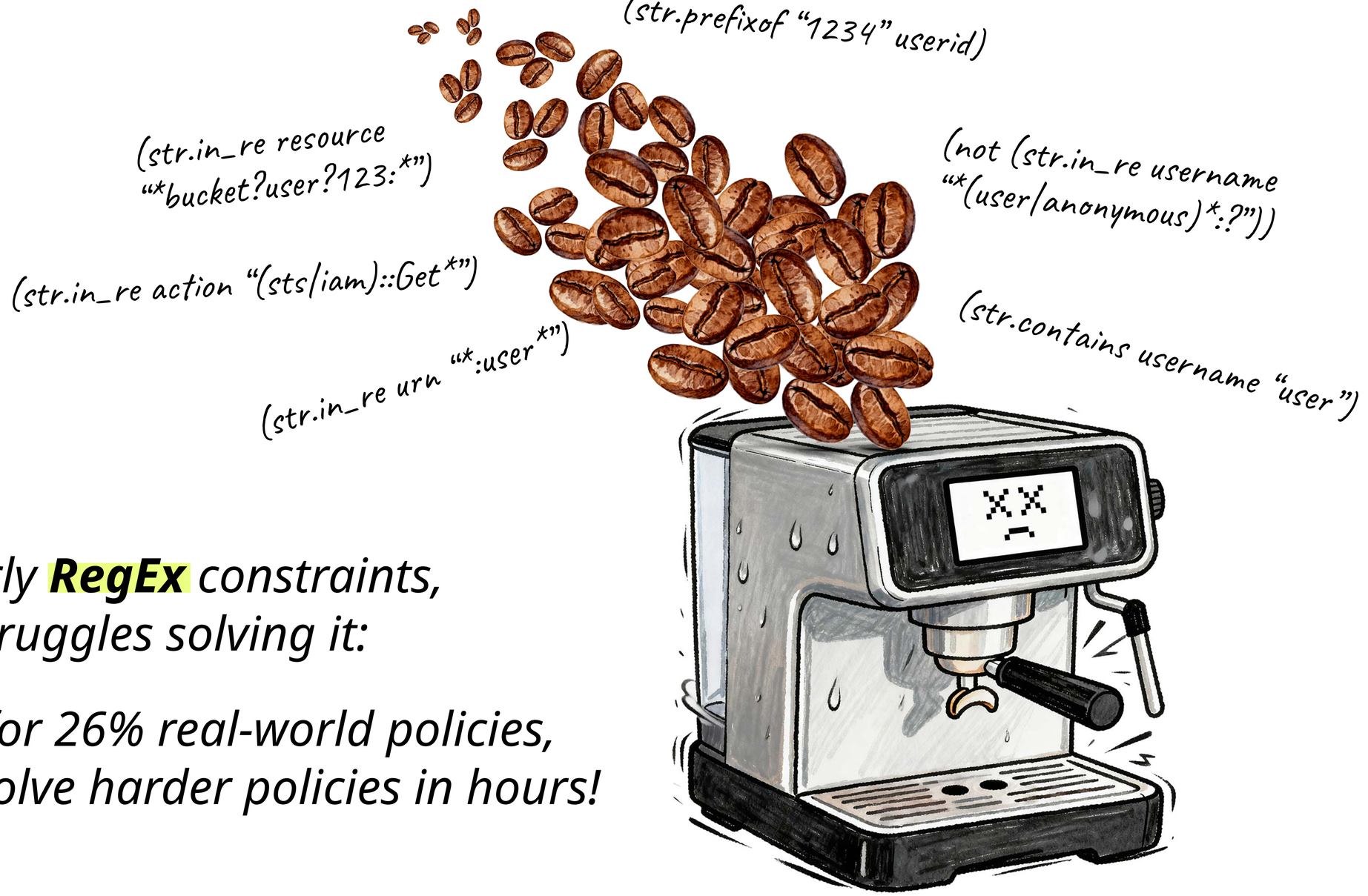
③

SMT Solver

④


Findings: AccessSummary [CAV '20]

# *However...*




*ACP are majorly **RegEx** constraints,*
*SMT solvers struggles solving it:*

*Z3 timed out for 26% real-world policies,*
*Cvc5 cannot solve harder policies in hours!*

# *Insight: Infinite Space of Strings*

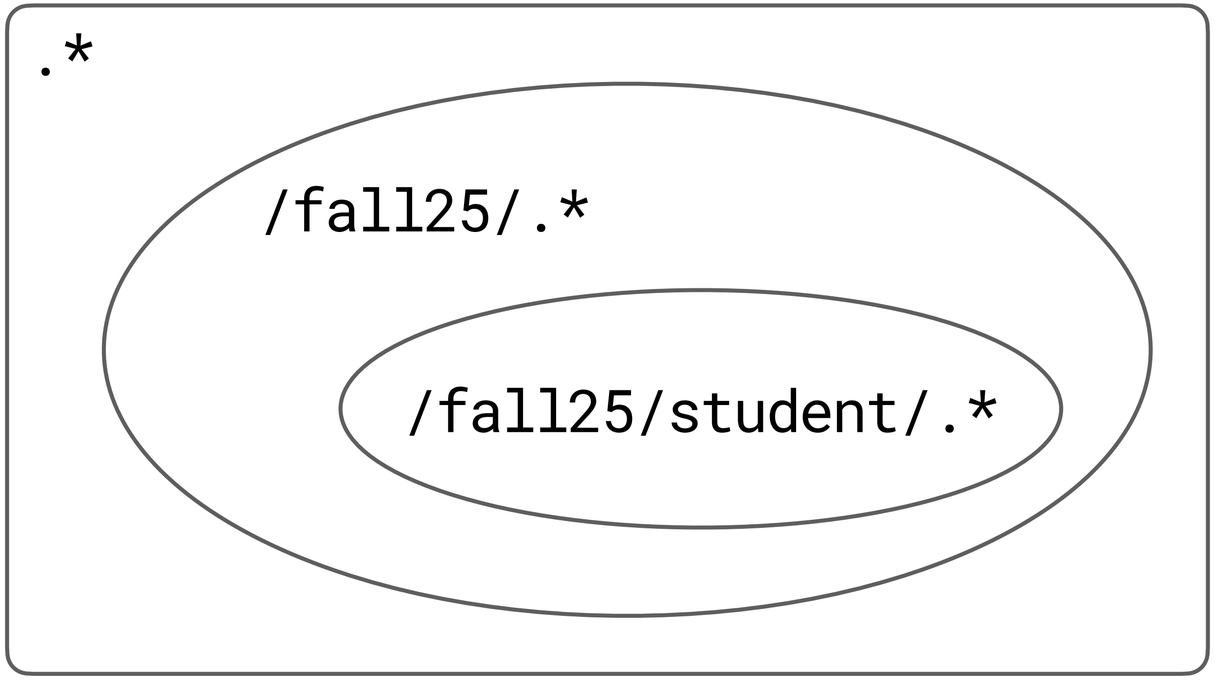
*Solvers "search" the **whole string space** to find answers, but the space is huge:*

*A paper strip with every ascii string **≤5 characters** circles around the Earth **5 times**!*

*Strings in Cloud Access Control Policies are typically **a lot longer**.*

# *Insight: Finite Space of String Equivalence Classes*

```
(assert (str.in_re user "/fall25/*"))
(assert (not
  (str.in_re user "/fall25/student/*")))
```




*If a variable **s** only has RegEx constraints,*

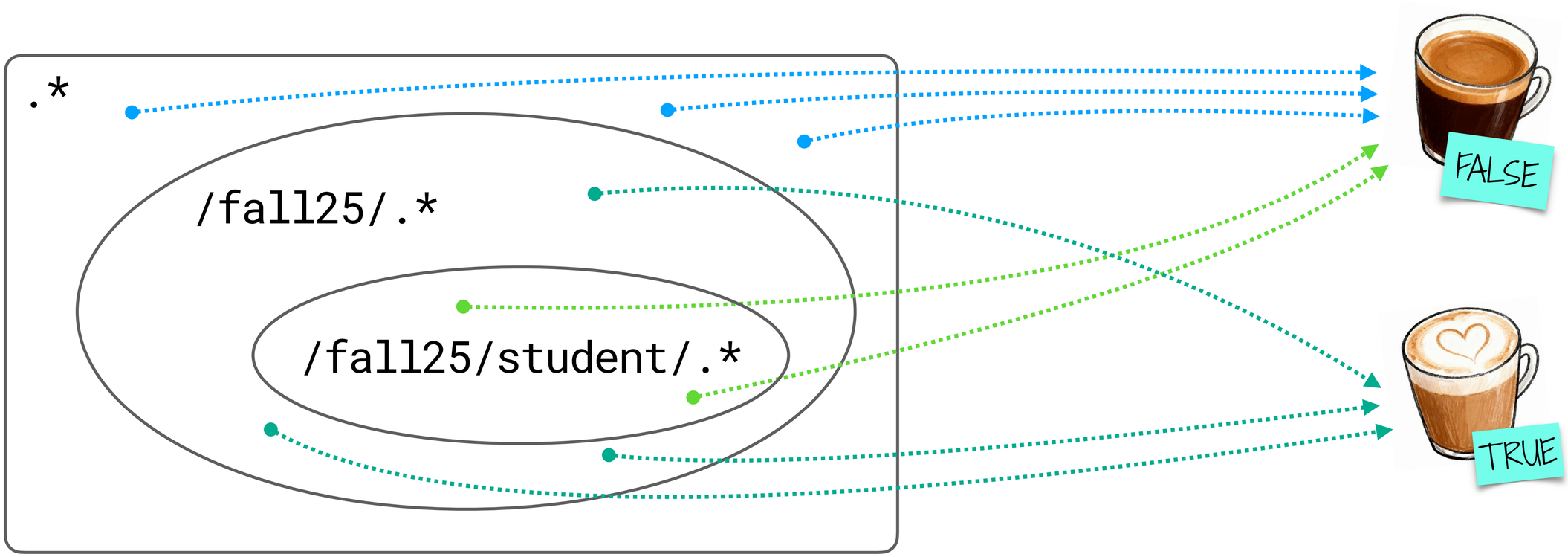
*Equivalence classes about the **RegExes that s belongs to** can be constructed.*

# *Insight: Finite Space of String Equivalence Classes*

```
(assert (str.in_re user "/fall25/*"))
(assert (not
  (str.in_re user "/fall25/student/*")))
```

***Same behavior***

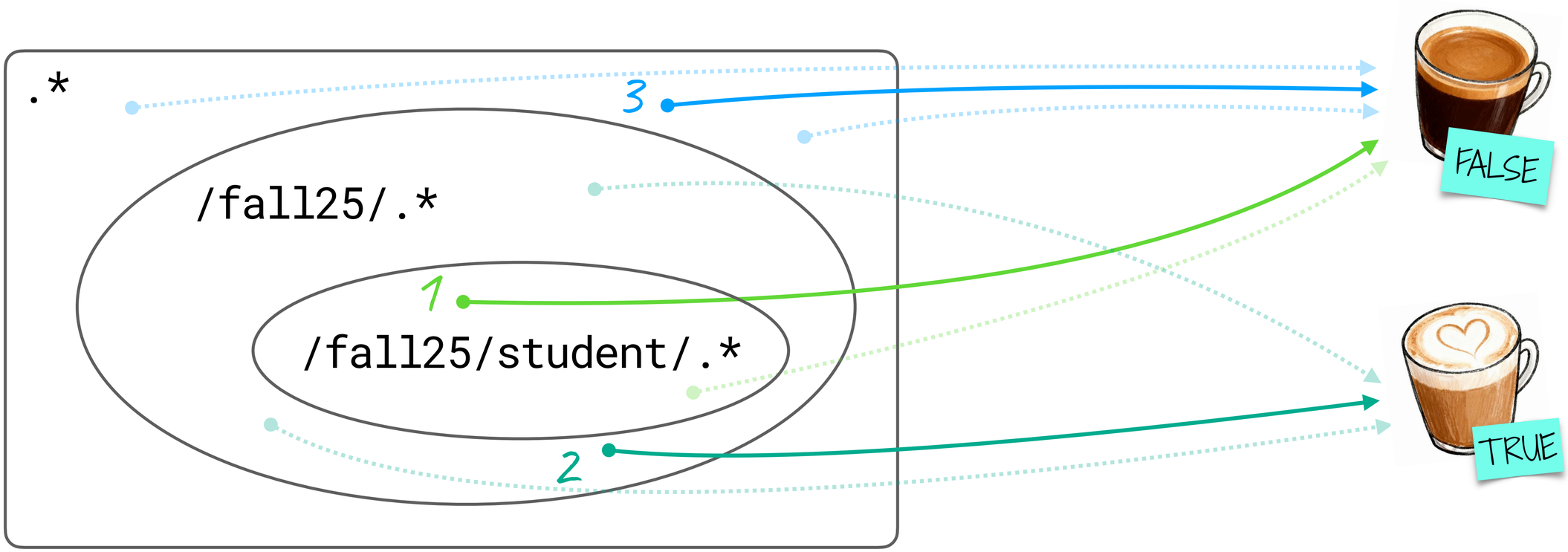
*in the same SEC!*



# *Insight: Finite Space of String Equivalence Classes*

```
(assert (str.in_re user "/fall25/*"))
(assert (not
  (str.in_re user "/fall25/student/*")))
```

Only consider the 3 SECs!



# Our Idea: RELIA

(str.in_re user "/fall25/.*")

(str.in_re user "/fall25/student.*")

**RE** *constraints over strings*

RELIA

**LIA** *constraints over* ***String Equivalence Classes***

(or (= user 1) (= user 2))

(= user 2)

(and (≥ user 1) (≤ user 3))

# RELIA: RegEx to LIA (Linear Integer Arithmetic)

**RE** *constraints over strings*

RELIA

***LIA** constraints over*
***String Equivalence Classes***

***SMT Solver***

# RELIA: RegEx to LIA (Linear Integer Arithmetic)

*ACP Analyzer*

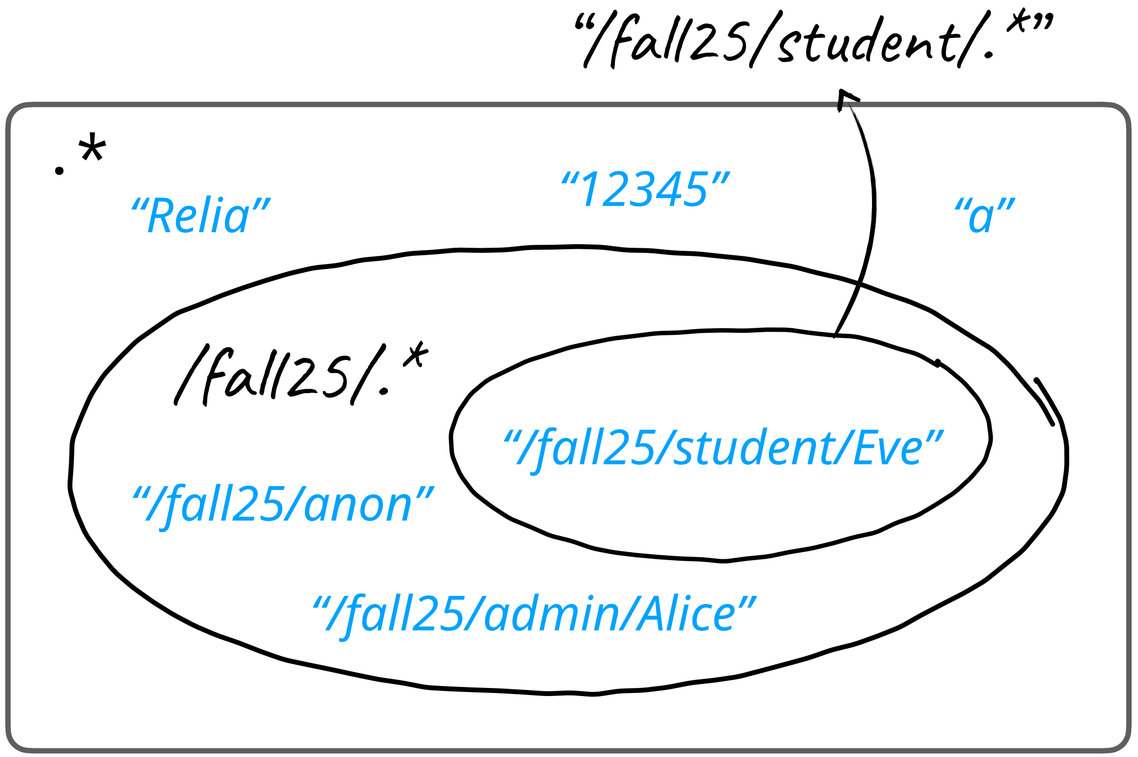
***RE** constraints over strings*

RELIA

***LIA** constraints over **String Equivalence Classes***

*SMT Solver*

# Compute SECs




**Constraints:** {"/fall25/.*",
"/fall25/student/.*"}

**SECs:** ① "/fall25/student/.*" ② "/fall25/¬(student/).*"

③ "¬(/fall25/).*"

# Convert Constraints

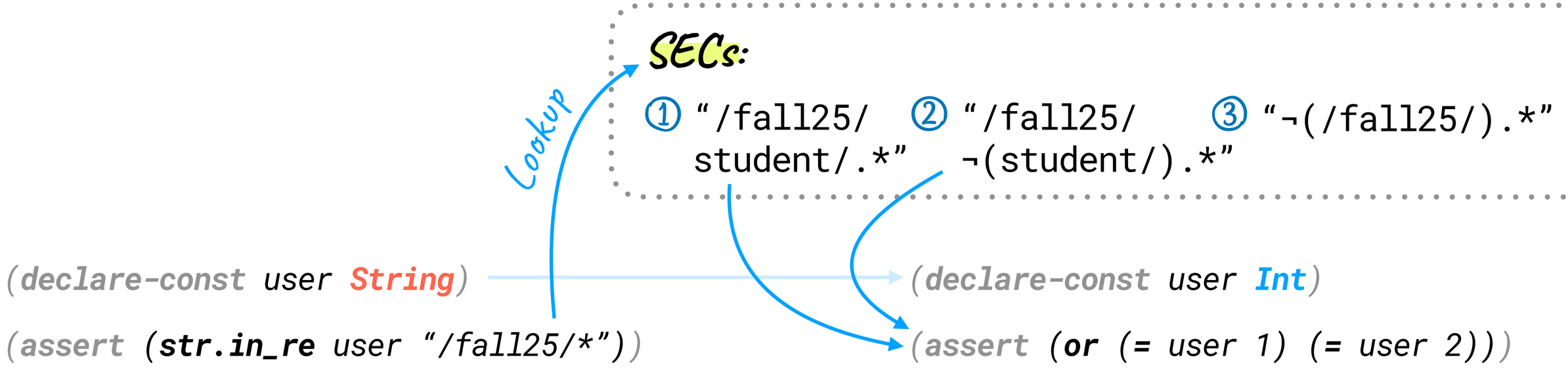
**SECs:**

① “/fall25/student/.*”

② “/fall25/¬(student/).*”

③ “¬(/fall25/).*”

# Convert Constraints



# Convert Constraints

**SECs:**

① “/fall25/student/.*”

② “/fall25/¬(student/).*”

③ “¬(/fall25/).*”

```
(declare-const user String)                    →  (declare-const user Int)
(assert (str.in_re user “/fall25/*”))          →  (assert (or (= user 1) (= user 2)))
(assert (not (str.in_re user                   →  (assert (not (= user 1))))
              “/fall25/student/*”)))
```

# Convert Constraints

SECs:

① “/fall25/student/.*”

② “/fall25/¬(student/).*”

③ “¬(/fall25/).*”

Add range

```
(declare-const user String)
(assert (str.in_re user “/fall25/*”))
(assert (not (str.in_re user
                “/fall25/student/*”)))
```

```
(declare-const user Int)
(assert (or (= user 1) (= user 2)))
(assert (not (= user 1)))
(assert (and
   (>= user 1) (<= user 3)))
```

# Convert Constraints

SECs:

① “/fall25/student/.*”

② “/fall25/¬(student/).*”

③ “¬(/fall25/).*”

```
(declare-const user String)
(assert (str.in_re user “/fall25/*”))
(assert (not (str.in_re user
                “/fall25/student/*”)))
```

```
(declare-const user Int)
(assert (or (= user 1) (= user 2)))
(assert (not (= user 1)))
(assert (and
   (>= user 1) (<= user 3)))
```

# *Handle Concatenation*

***id*** = “/“ + ***semester*** + “/” + ***role***

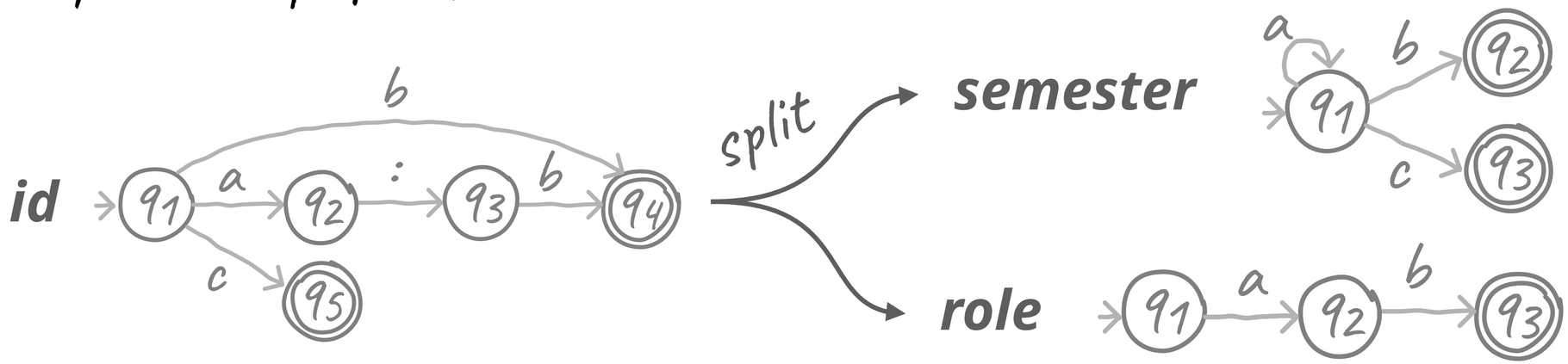
***id*** ∈ “/(fall.*|spring.*)/
(admin|professor)”

*semester* ∈ ???

*role* ∈ ???

*We use a* ***novel FA algorithm*** *to deal with* ***string concatenations****.*

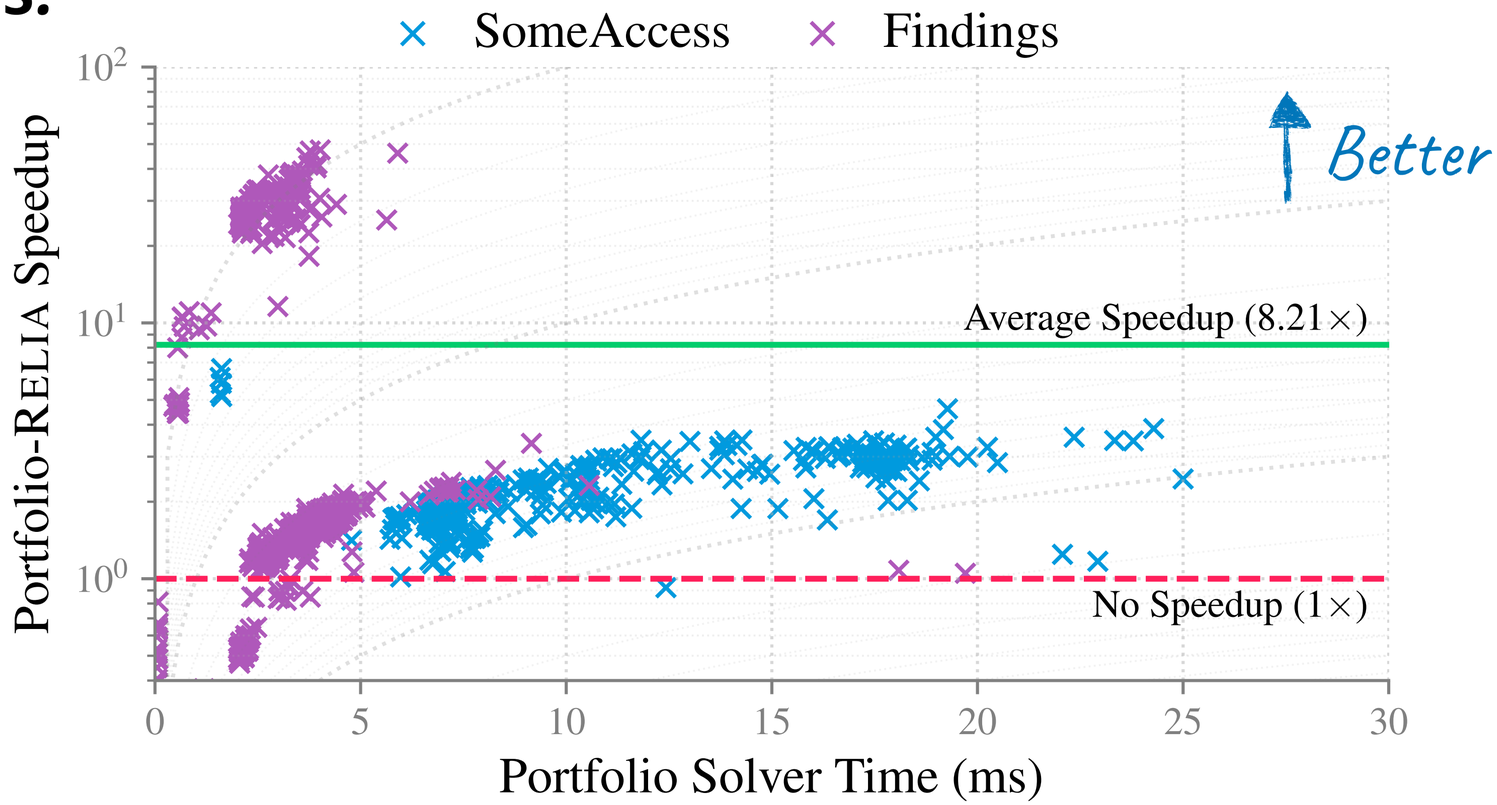
*Go to our poster / paper for more!*




**not the actual algorithm, for illustration only*

# *Evaluations*

## *Real policies:*




*Speed up the analysis of real policies from **Huawei Cloud** tenants by **8.21x***

# *Evaluations*

## *Hard policies:*

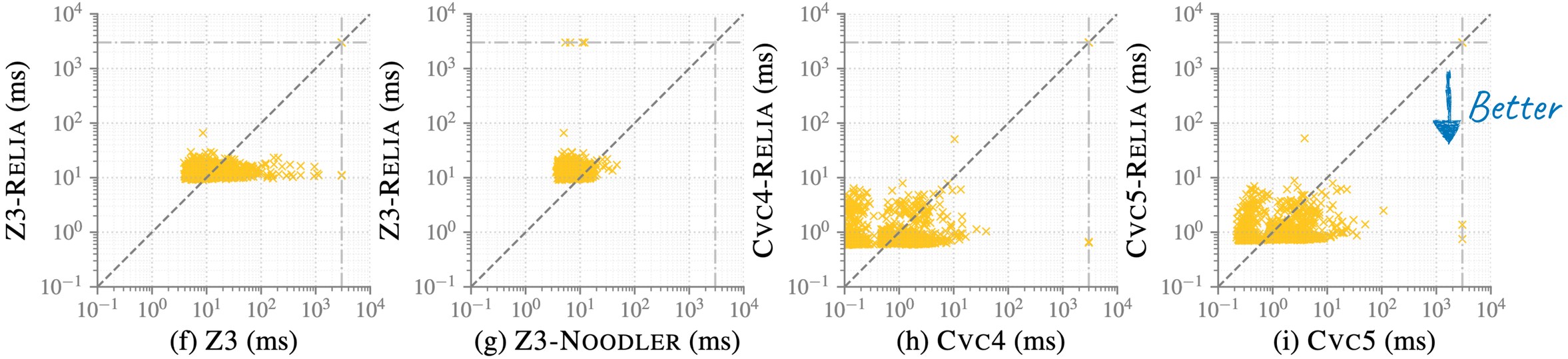
| *#Cases solved in time* | *Z3* | *Cvc4* | *Cvc5* | *Z3-Noodler* | *Cvc5-RELIA* |
|---|---|---|---|---|---|
| 10ms | 0 | 0 | 0 | 0 | **1** |
| 100ms | 0 | 0 | 0 | 2 | **4** |
| 1s | 0 | 0 | 0 | 5 | **7** |
| 10s | 1 | 0 | 0 | 5 | **8** |

*Solve **8/10** hard policies in seconds while Z3 / Cvc5 cannot solve most in hours!*

***View our dataset:*** *https://zenodo.org/records/17236980*

# *Evaluations*

## *We can also accelerate some general SMT string solving!*




*View more results in our poster / paper!*

# Conclusion

1. ***Access Control Policies*** *are essential in Cloud, but analyzing them are slow due to difficulty of solving* ***string constraints*** *in SMT.*

2. *We propose* **RELIA**, *a tool to bypass string solving by utilizing* ***String Equivalence Classes***.

3. **RELIA** *can accelerate* ***ACP*** *analyzers along with some* ***general*** *SMT string problems.*

# Happy to take your questions!

## RELIA: Accelerating the Analysis of Cloud Access Control Policies

*NetVerify*, ANTS, Xi'an Jiaotong University; **Huawei Cloud**

**Illustrations in the slides are generated using Doubao AI.*

ASE 2025 *the* 40th IEEE/ACM International Conference on **Automated Software Engineering (ASE 2025)** *Seoul, Korea* *November 16 - 20, 2025*