IEEE/ACM International Symposium on Quality of Service
2–4 July 2025 // Gold Coast, Australia

LEOPARD: Accelerating Cloud-based Access Control Policy Verification Using Logical Encoding Optimization

Xing Fang[1], Feiyan Ding[1], Mingyuan Song[1], Yuntao Zhao[1], Lizhao You[1],
Qiao Xiang[1], Linghe Kong[2], Jiwu Shu[1, 3] , Xue Liu[4],

[1] Xiamen University, [2] Shanghai Jiao Tong University, [3] Minjiang University , [4]McGill University

2025/07/03

Access Control Policy Configuration Errors Are Common

McGraw Hill's S3 bu students' grades ar

Educator gets an F for security

Misconfigured Azure Blob Storage Exposed the Data of 65K Companies and 548K Users

According to SOCRadar, "the amount and scale of the leaked data make it the most significant B2B data leak in the recent history of cybersecurity."

October 20, 2022

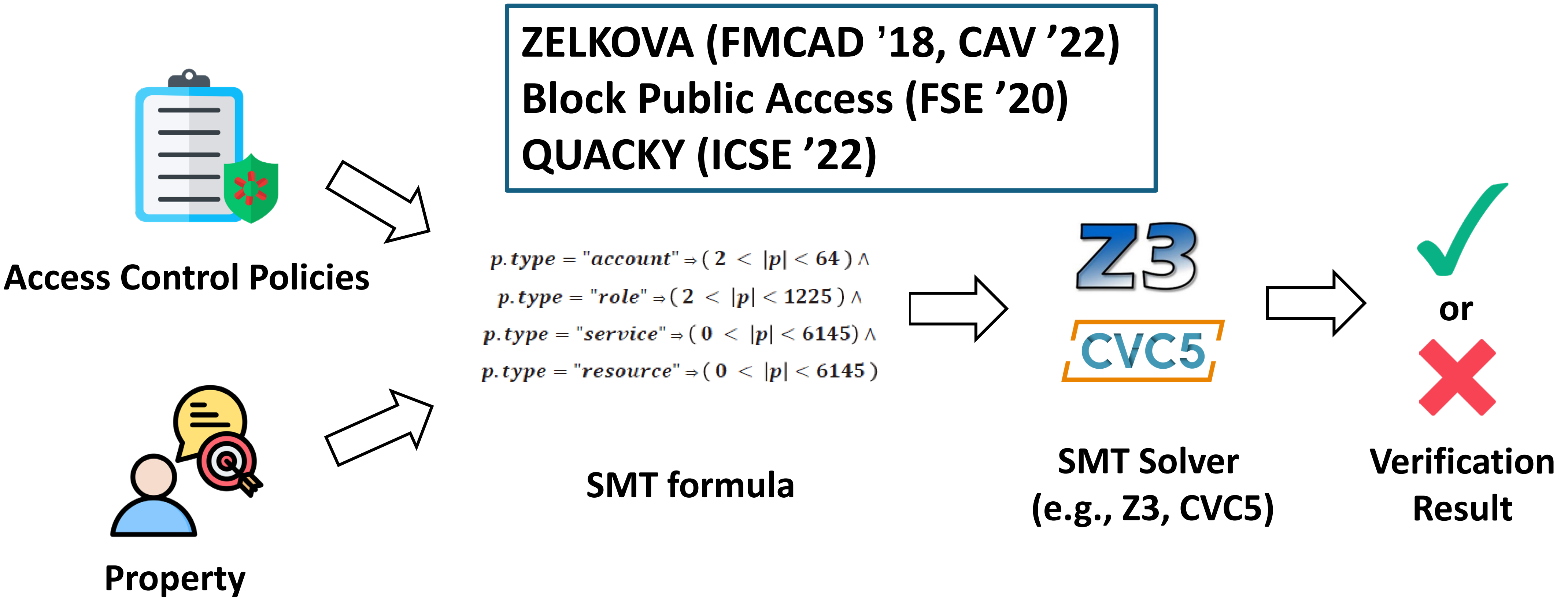
Salesforce cloud outage caused by security change

By Richard Chirgwin

Sep 27 2023 12:34PM

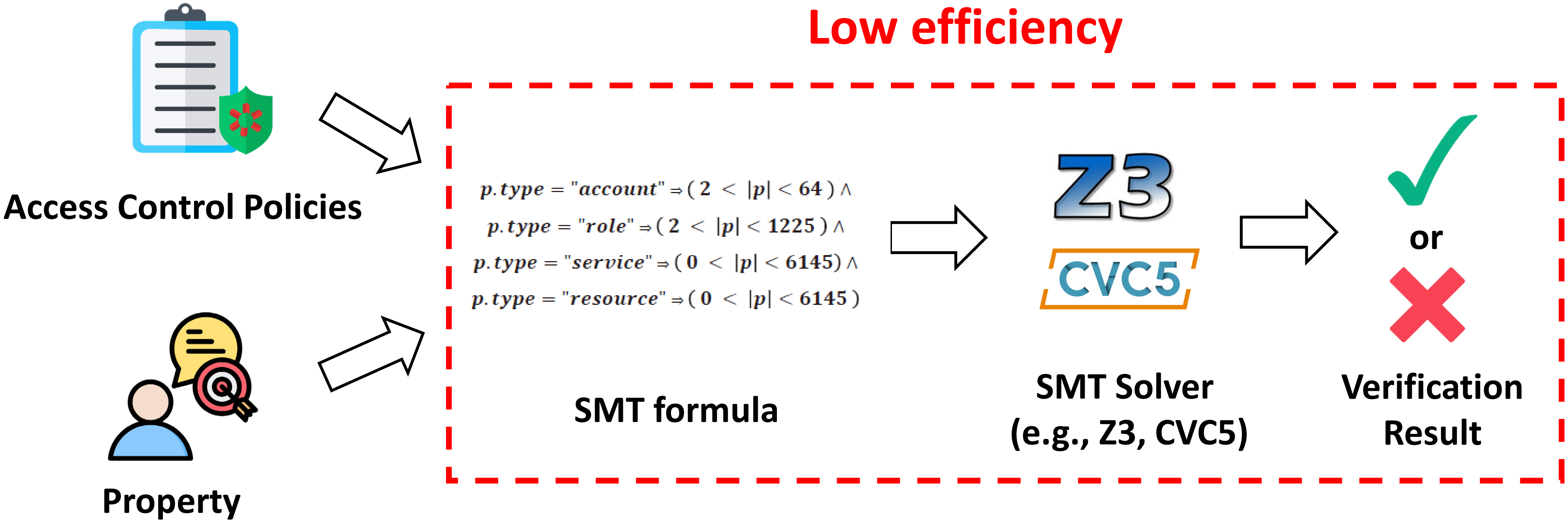
Alibaba Cloud outage sees apps taken offline

Policy Verification Helps Prevent Misconfigurations



Existing works use *SMT-based* verification to check whether access control policies satisfy desired properties.

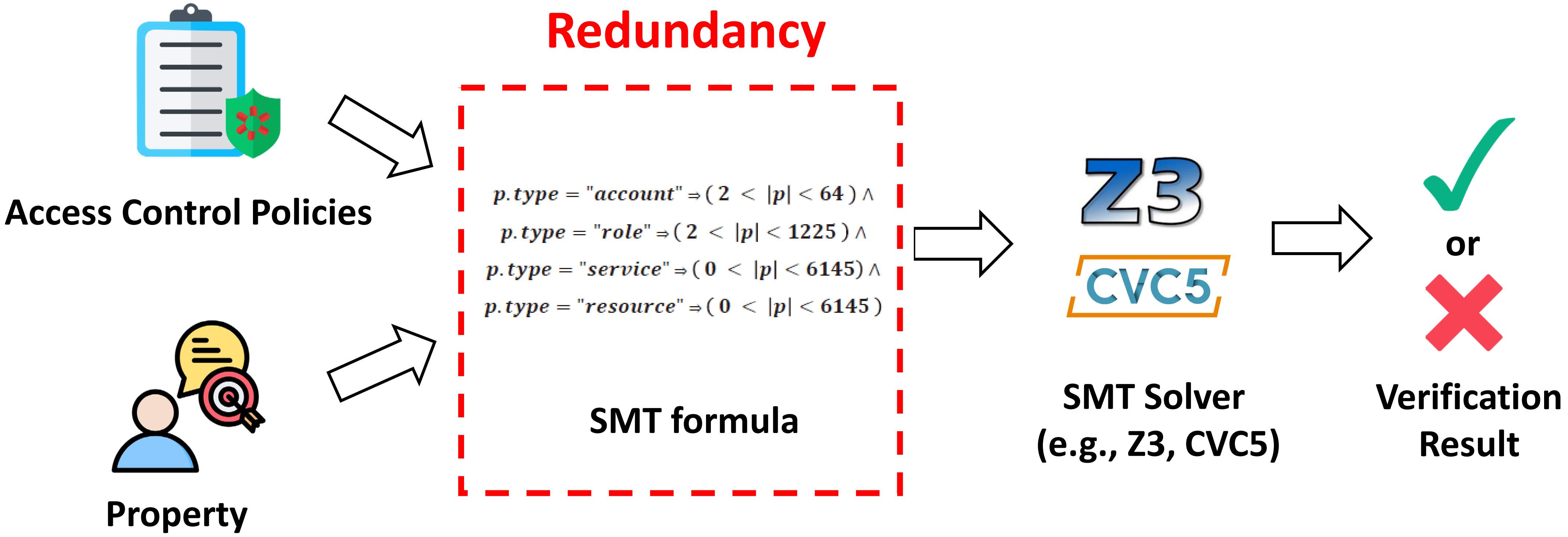
Challenge: SMT Verification at Scale



Existing tools lack the efficiency needed to support billions of SMT queries per day[1].

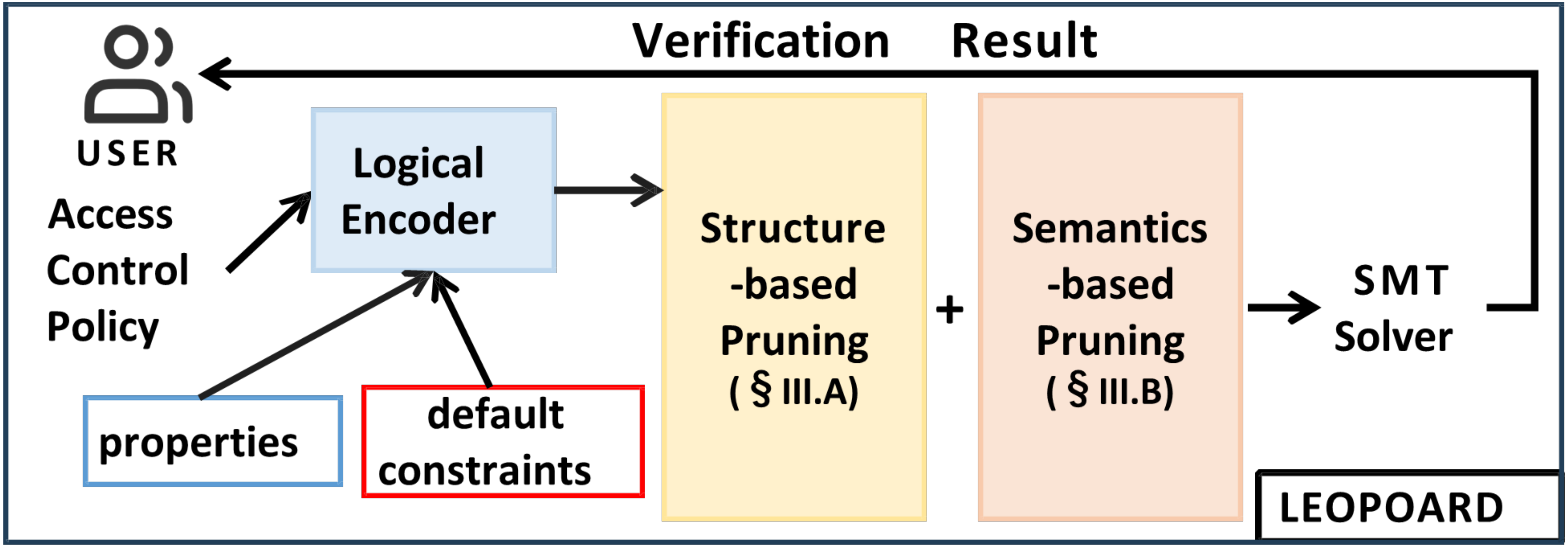
[1] Rungta, N. "A Billion SMT Queries a Day (Invited Paper)". CAV'22

Motivation: SMT Formula Encoding Redundancy



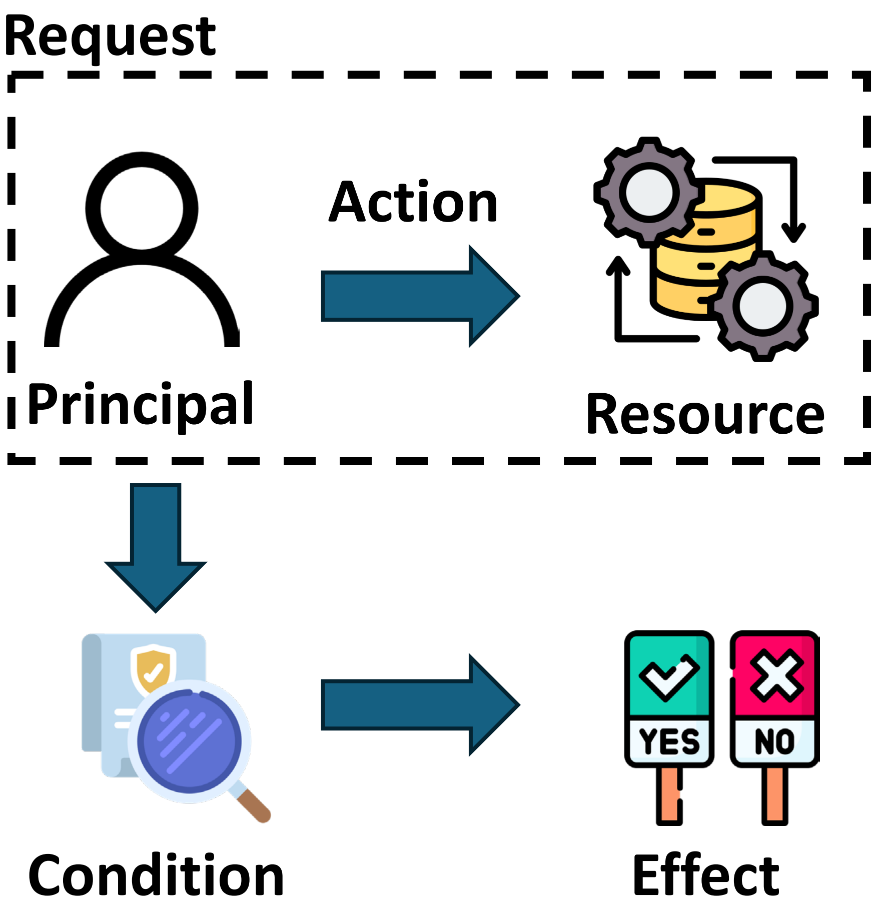
Existing works *encode redundant constraints* without considering their relevance to the verification property.

Our Approach: LEOPARD



Optimizing logical encoding to accelerate policy verification.

Cloud-based Access Control Policy



- **Effect**: **Allow or Deny** the request
- **Principal**: **Who** makes the request
- **Action**: **What operation** is performed
- **Resource: Which resource** is targeted
- **Condition**: Under **what conditions** it applies

Access control policies define who can perform which actions on which resources.

Cloud-based Access Control Policy

```
{(Allow
  Principal :   "*",
  Action    :   "GetObject",
  Resource  :   "examplebucket/*",
  Condition:    ""),
 (Deny
  Principal :   "*",
  Action    :   "GetObject",
  Resource  :   "examplebucket/*",
  Condition:    (StringNotLike, UserId, EXAMPLEID:*))}
```

Policy X:

Allow all users to get objects from the bucket

Deny users whose UserId does not match EXAMPLEID:* to get objects from the bucket

```
{(Allow
  Principal :   "*",
  Action    :   "*",
  Resource  :   "examplebucket/*",
  Condition:    "")
```

Policy Y:

Allow all users to perform all actions on the bucket

Policy Example

Cloud-based Access Control Policy

Actions	Resources
GetObject	examplebucket/*
GetBucketPolicy	
PutBucketAcl	
PutBucketCors	

Action Table Example

String	Length
UserId	(0, 64)
UserName	(0, 64)
AgencyId	(0, 64)
AgencyName	(0, 64)

String Table Example

Default constraints define which requests are considered valid by the cloud platform.

Logical Encoding for Access Control Policy

$$[P] = \left(\bigvee_{S \in Allow} [S] \right) \wedge \neg \left(\bigvee_{S \in Deny} [S] \right)$$

Policy Encoding:

A request is allowed if it satisfies any Allow statement and does not match any Deny statement.

$$[S] = \left(\bigvee_{v \in S(P)} p = v \right) \wedge \left(\bigvee_{v \in S(A)} a = v \right) \wedge \left(\bigvee_{v \in S(R)} r = v \right) \wedge \left(\bigwedge_{O \in S(C)} O \right).$$

Statement Encoding:

A request matches a statement if the principal, action, and resource each match, and all conditions are satisfied.

Logical Encoding for Access Control Policy

```
{(Allow
  Principal :  "*",
  Action    :  "GetObject",
  Resource  :  "examplebucket/*",
  Condition:   ""),
 (Deny
  Principal :  "*",
  Action    :  "GetObject",
  Resource  :  "examplebucket/*",
  Condition:   (StringNotLike, UserId, EXAMPLEID:*))}
```

Policy X

$$a = \text{“}GetObject\text{”} \land$$
$$\text{"examplebucket/"}\ prefixOf\ r \land$$
$$\neg(a = \text{“}GetObject\text{”} \land$$
$$\text{"examplebucket/"}\ prefixOf\ r \land$$
$$\land \neg(\text{"}EXAMPLEID\text{: "}\ prefixOf\ UserId))$$

SMT encoding of Policy X

```
{(Allow
  Principal :  "*",
  Action    :  "*",
  Resource  :  "examplebucket/*",
  Condition:   "")
```

Policy Y

$$\text{"examplebucket/"}\ prefixOf\ r$$

SMT encoding of Policy Y

Logical Encoding for Default Constraints

$$[D_A] = \bigvee_{(T_A,T_R)\in ActionTable} \left(\left(\bigvee_{v\in T_A} a = v \right) \wedge \left(\bigvee_{v\in T_R} r = v \right) \right)$$

Action Table Encoding:

An action–resource pair must match an entry in the Action Table.

$$[D_S] = \bigwedge_{(s,min,max)\in StringTable} min \leq |s| \leq max$$

String Table Encoding:

A string field must satisfy its type-specific length range.

Logical Encoding for Default Constraints

Actions	Resources
GetObject	examplebucket/*
GetBucketPolicy	
PutBucketAcl	
PutBucketCors	

Action Table Example

$$
\begin{aligned}
&((a = \text{“}GetObject\text{”} \lor \\
&a = \text{"}GetBucketPolicy\text{"} \lor \\
&a = \text{“}PutBucketAcl\text{”} \lor \\
&a = \text{"}PutBucketCors\text{"}\ ) \land \\
&\text{“}examplebucket/\text{”}\ prefixOf\ r)
\end{aligned}
$$

SMT encoding of Action Table

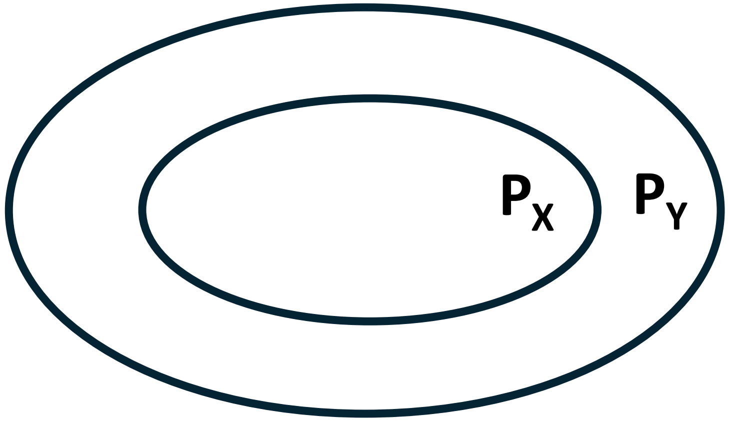
String	Length
UserId	(0, 64)
UserName	(0, 64)
AgencyId	(0, 64)
AgencyName	(0, 64)

String Table Example

$$
\begin{aligned}
&(\ 0\ < \ |UserId| < 64\ ) \land \\
&(\ 0\ < \ |UserName| < 64\ ) \land \\
&(\ 0\ < \ |AgencyId| < 64) \land \\
&(\ 0\ < \ |AgencyName| < 64)
\end{aligned}
$$

SMT encoding of String Table

Logical Encoding for Verification Problem



Property: $\mathbf{P_X \Rightarrow P_Y}$
(i.e., all requests allowed by P_X must be allowed by P_Y)

P_Y is the trusted zone → **Safety:** ensure P_X does not allow any untrusted request

P_X is the trusted zone → **Availability:** ensure P_Y does not deny any legitimate request

SMT formula: $([P_X] \land \neg[P_Y] \land [D])$

Result: If unsatisfiable, the property holds: $P_X \Rightarrow P_Y$.

Observation: Redundancy in Default Constraint Encoding

Property: $[P_X] \land \neg[P_Y]$

Default Constraint: $[D]$

Encode **all** constraints

SMT formula: $([P_X] \land \neg[P_Y] \land [D])$

Existing works encode redundant constraints into the SMT formula.

Structural Redundancy

$$a = \text{“}GetObject\text{”} \wedge$$
$$\text{"examplebucket/"}\ prefixOf\ r \wedge$$
$$\neg(a = \text{“}GetObject\text{”} \wedge$$
$$\text{"examplebucket/"}\ prefixOf\ r \wedge$$
$$\wedge \neg(\text{"}EXAMPLEID\text{:"}\ prefixOf\ UserId))$$

SMT encoding of Policy X

$$\text{"examplebucket/"}\ prefixOf\ r$$

SMT encoding of Policy Y

Policy encoding

$$(a = \text{“}GetObject\text{”} \vee$$
$$a = \text{"}GetBucketPolicy\text{"} \vee$$
$$a = \text{“}PutBucketAcl\text{”} \vee$$
$$a = \text{"}PutBucketCors\text{"}) \wedge$$
$$\text{“}examplebucket/\text{”}\ prefixOf\ r) \wedge$$
$$(0 < |UserId| < 64) \wedge$$
$$(0 < |UserName| < 64) \wedge$$
$$(0 < |AgencyId| < 64) \wedge$$
$$(0 < |AgencyName| < 64)$$

Default constraints encoding

Constraints are structurally redundant if they have no dependency on variables used in the verification property.

Semantic Redundancy

$$a = \text{“}GetObject\text{”} \wedge$$
$$\text{"examplebucket/"}\ prefixOf\ r \wedge$$
$$\neg(a = \text{“}GetObject\text{”} \wedge$$
$$\text{"examplebucket/"}\ prefixOf\ r \wedge$$
$$\wedge \neg(\text{"}EXAMPLEID\text{:"}\ prefixOf\ UserId))$$

SMT encoding of Policy X

$$\text{"examplebucket/"}\ prefixOf\ r$$

SMT encoding of Policy Y

Policy encoding

$$(a = \text{“}GetObject\text{”} \vee$$
$$a = \text{"}GetBucketPolicy\text{"} \vee$$
$$a = \text{“}PutBucketAcl\text{”} \vee$$
$$a = \text{"}PutBucketCors\text{"}) \wedge$$
$$\text{“}examplebucket/\text{”}\ prefixOf\ r) \wedge$$
$$(0 < |UserId| < 64) \wedge$$
$$(0 < |UserName| < 64) \wedge$$
$$(0 < |AgencyId| < 64) \wedge$$
$$(0 < |AgencyName| < 64)$$

Default constraints encoding

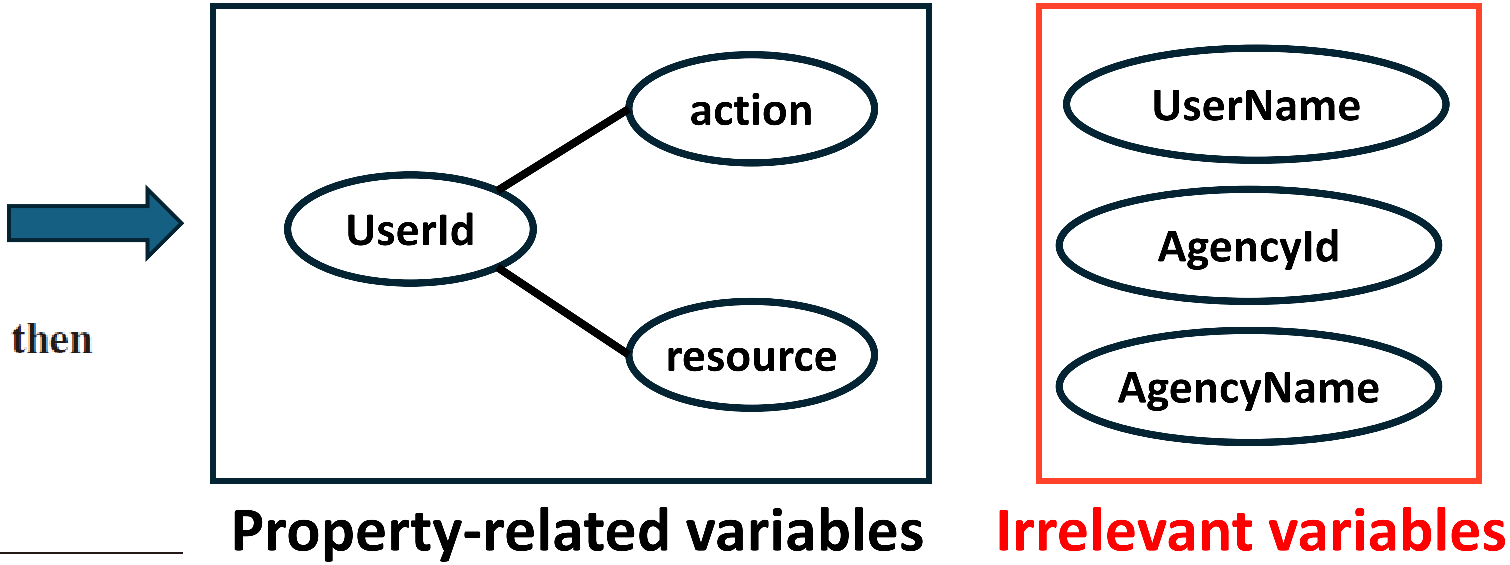
Constraints are semantically redundant if they are unsatisfiable under the variable values fixed by the verification property.

Structure-based Pruning

Algorithm 1 StrucutreBasedPruning(D, Prop)

Input: Formulas D, $Prop$
Output: Pruned formulas D_p

Initialize a queue Q
Q.enqueue($Prop$)
$D_p = \{\emptyset\}$
while NotEmpty(Q) **do**
    $f = Q$.dequeue()
    for formula d in D **do**
        if d and f share a variable and $d \notin D_p$ **then**
            Q.enqueue(d)
            $D_p = D_p \cup d$
return D_p



Perform dependency analysis to prune default constraints that have no dependency on variables used in the verification property.

Semantic-based Pruning

Algorithm 2 SemanticsBasedPruning(D, Prop)

Input: Formulas D_A, $Prop$
Output: pruned formulas D_p

$D_p = \{\emptyset\}$
for formula d in D_A **do**
    $intersect = d.actions \cap Prop.allowedActions$
    if $intersect \neq \emptyset$ **then**
        $d.actions = intersect$
        $D_p = D_p \cup d$
return D_p

$a =$ "$GetObject$"

Allowed actions

$a =$ "$GetBucketPolicy$"

$a =$ "$PutBucketAcl$"

$a =$ "$PutBucketCors$"

Semantically invalid actions

Remove actions that are semantically invalid under the property's constraints.

Combined Pruning Yields Significant Simplification

$$(a = \text{"GetObject"} \lor$$
$$a = \text{"GetBucketPolicy"} \lor$$
$$a = \text{"PutBucketAcl"} \lor$$
$$a = \text{"PutBucketCors"}) \land$$
$$\text{"examplebucket/"}\ prefixOf\ r) \land$$
$$(0 < |UserId| < 64) \land$$
$$(0 < |UserName| < 64) \land$$
$$(0 < |AgencyId| < 64) \land$$
$$(0 < |AgencyName| < 64)$$

Structure-based Pruning

Semantic-based Pruning

$$a = \text{"GetObject"} \land$$
$$\text{"examplebucket/"}\ prefixOf\ r) \land$$
$$(0 < |UserId| < 64)$$

The SMT formula is significantly simplified by pruning redundant constraints, while preserving correctness.

Evaluation

- Experiment Setting
  - **Datasets:** 2 synthesized datasets from AWS and a large cloud provider.
  - **Properties**: binary and quantitative analysis.
  - **Metric**: solving time before and after pruning.
  - **Baseline**: advanced SMT solvers (Z3, Z3str3RE, OSTRICH, ABC) and the state-of-the-art tool QUACKY for quantitative analysis.

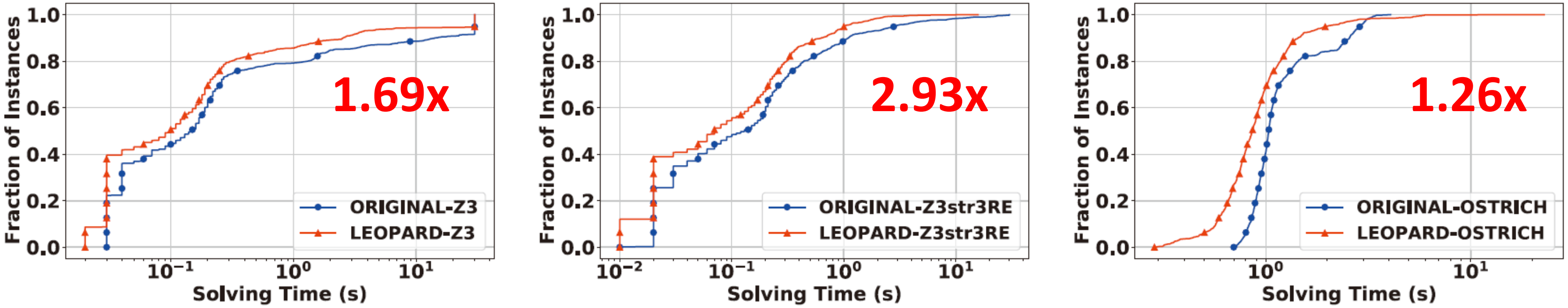
Evaluation

TABLE I: The Number of the Original and Pruned Instances on Datasets.

Dataset	**#Instance**		
	ORIGINAL	**LEOPARD**	**RATIO(%)**
Dataset1	2792	1267	45%
Dataset2	3018	1836	61%

A significant portion of formulas can be pruned: 45% in Dataset1 and 61% in Dataset2.

Evaluation



(a) Z3 (b) Z3str3RE (c) OSTRICH

Fig. 5: CDF of solving times for binary analysis of original and pruned SMT instances in Dataset1 across three solvers

Consistent Speedup

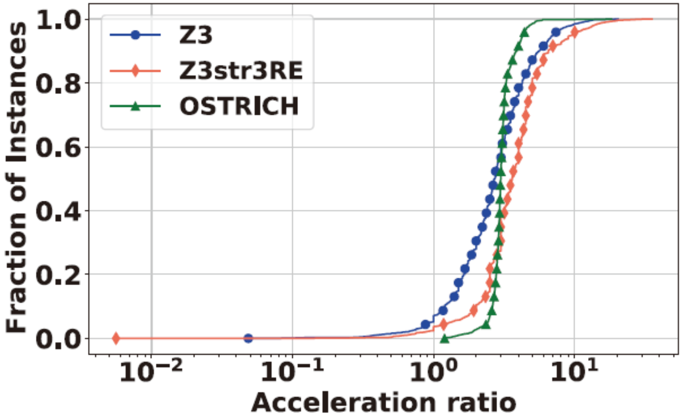


Fig. 6: The ratio of solving time for binary analysis of the original and pruned instances on Dataset1 for all solvers

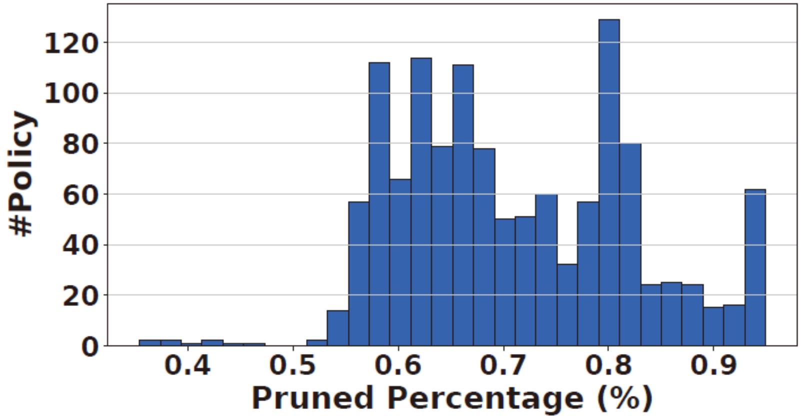
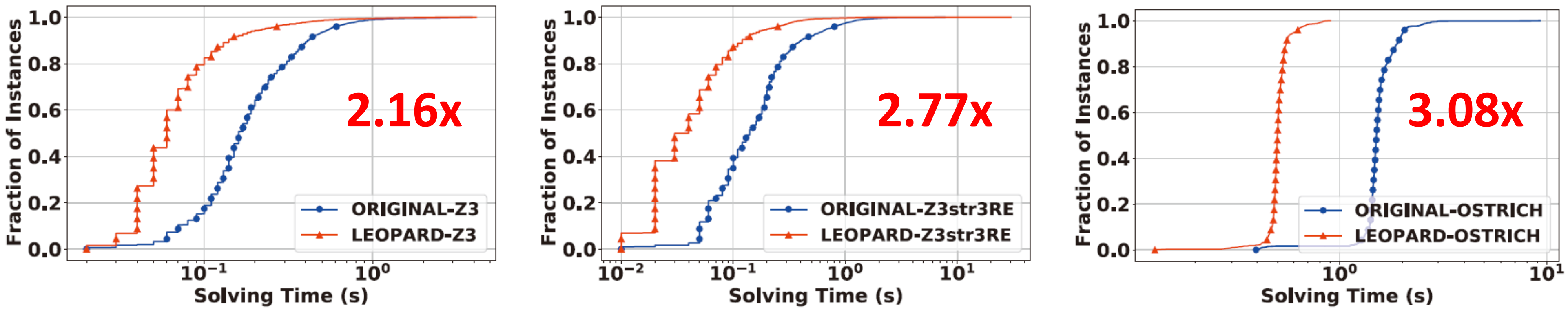


Fig. 7: The distribution of the percentage of atomic formulas pruned by LEOPARD across all pruned instances in Dataset1

Significant Pruning

Dataset1 binary analysis

Evaluation



(a) Z3 (b) Z3str3RE (c) OSTRICH

Fig. 8: CDF of solving times for binary analysis of original and pruned SMT instances in Dataset2 across three solvers

Consistent Speedup

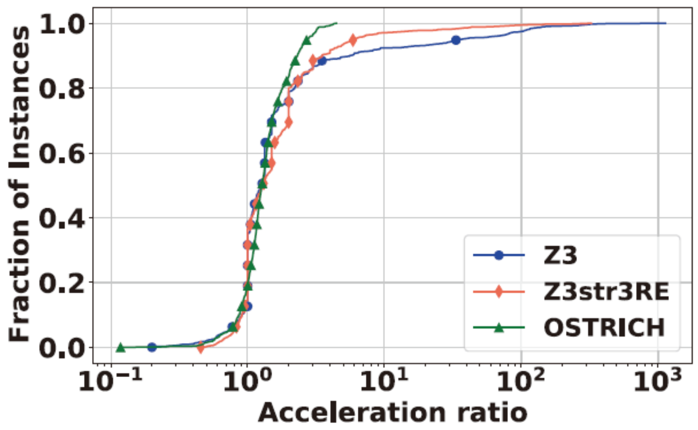


Fig. 9: The ratio of solving time for binary analysis of the original and pruned instances on Dataset2 for all solvers

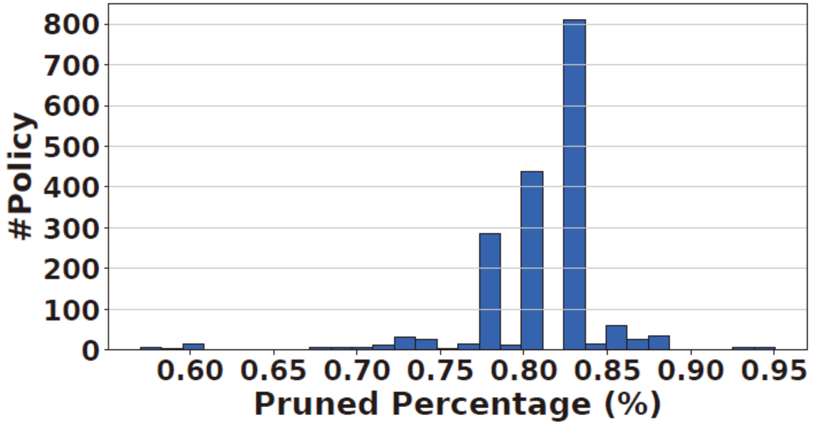


Fig. 10: The distribution of the percentage of atomic formulas pruned by LEOPARD across all pruned instances in Dataset2

Significant Pruning

Dataset2 binary analysis

Evaluation

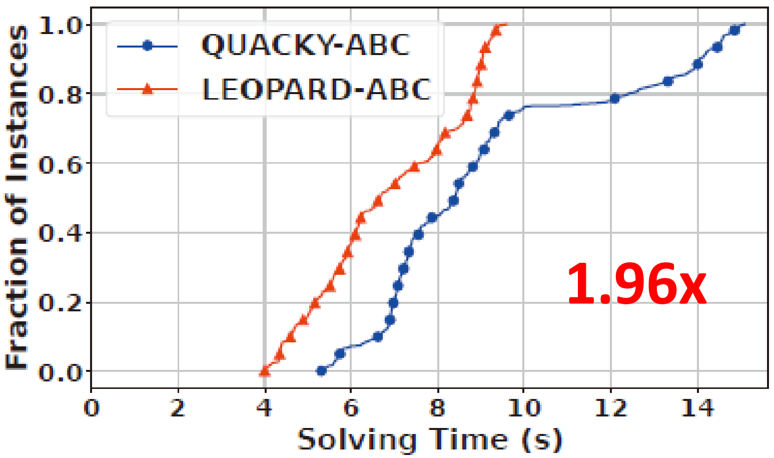


Fig. 11: CDF of solving times for quantitative analysis of original and pruned SMT instances in Dataset1 using ABC

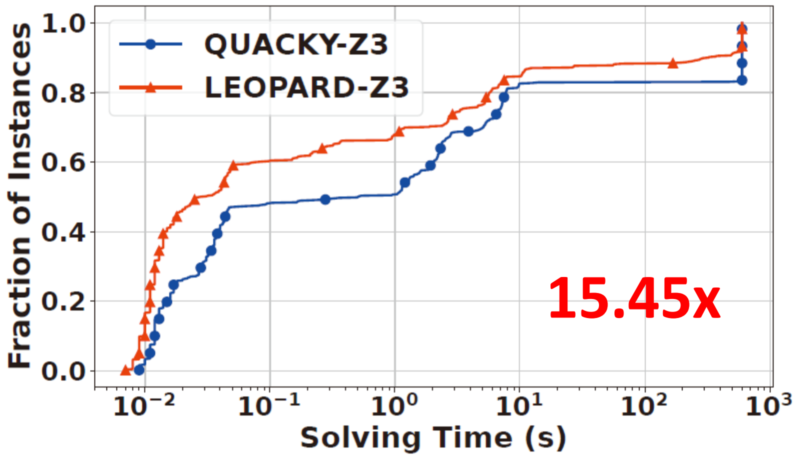


Fig. 12: CDF of solving times for quantitative analysis of original and pruned SMT instances in Dataset1 using Z3

Dataset1 quantitative analysis

Conclusion

- LEOPARD: a method to accelerate access control policy verification via logical encoding optimization
  - Structure-based Pruning
  - Semantic-based Pruning
- Extensive evaluation