Symbolic Boolean Derivatives
aka: Regular Expression solving in Z3

Caleb Stanford, Margus Veanes, and Nikolaj Bjørner
Raw String Data in Security-Critical Software

Strings: The Universal Datatype

serialize: Object -> String;

JSON
CSV
XML
YAML
TOML
...

OBJECT -> STREAM OF BYTES -> FILE -> DB -> MEMORY
Raw String Data in Security-Critical Software

Top web application vulnerabilities (OWASP)

- A6 — Security Misconfiguration (84%)
- A7 — Cross-Site Scripting (XSS) (53%)
- A2 — Broken Authentication (45%)
- A5 — Broken Access Control (37%)
- A1 — Injection (29%)
- A9 — Using Components with Known Vulnerabilities (13%)
- A3 — Sensitive Data Exposure (13%)
- A4 — XML External Entities (XXE) (5%)
# Raw String Data in Security-Critical Software

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
</table>

Regular expression Controls access to secure resource!
Verification using SMT

String and Regex Solvers

Z3
CVC4
Norn (2015)
Z3str2 (2017)
Z3str3 (2017)
Sloth (2017)
Ostrich (2019)
Trau (2019)
Z3-Trau (2020)
...

Industrial applications

"The total number of invocations of Zelkova ranges from a few million to tens of millions in a single day"
Problem: **Mismatch** Between Real-World Regex Constraints and Solvers

Real-world constraints:

\[
date \in \text{\textbackslash d}\{4\}-[^{a-zA-Z}\}\{3\}-\text{\textbackslash d}\{2\} \land (date \in 2019.* \lor date \in 2020.*).
\]

- Symbolic (e.g. Unicode)
- Boolean combinations

Solvers and theory:

(ab | aba)* | abb

- Finite alphabet
- Union, concat, and *
Problem: **Mismatch** Between Real-World Regex Constraints and Solvers

Real-world constraints:

\[
date \in \d{4}-[a-zA-Z]\{3\}-\d{2} \land \\
(date \in 2019.* \lor date \in 2020.*).
\]

- Symbolic (e.g. Unicode)
- Boolean combinations

Solvers and theory:

(ab | aba)* | abb

- Finite alphabet
- Union, concat, and *
A New Regex Solver: dZ3

Internship project: summer 2020

Now the default solver in Z3: https://github.com/Z3Prover/z3

Theory: Generalization of “derivatives”:
- Symbolic (e.g. Unicode)
- Boolean combinations
Outline

• Motivating example

• Technical challenges
  • Boolean: Handling complement/intersection
  • Symbolic: Large character space

• Experimental results
Cloud Security Policies

Major cloud providers

Security policy configuration languages

{ "Effect": "Allow",  
  "NotPrincipal": { "AWS": "111122223333" },  
  "Action": "*",  
  "Resource": "arn:aws:s3:::test-bucket"  
} ...

"if": { "allOf": [  
  { "field": "type",  
    "equals": "Service.Web/sites"  
  },  
  {  
    "field": "kind",  
    "like": "*api"  
  }  
],  
"bindings": [  
  { "role": "roles/resourcemanager.orgAdmin",  
    "members": [  
      "user:mike@example.com",  
      "group:admins@example.com",  
      "domain:google.com"  
    ]  
  }  
} ...
Anatomy of a Security Policy

API App should only be accessible over HTTPS

"if": { "allOf": [ {
  "field": "type",
  "equals": "Service.Web/sites"
}, {
  "field": "kind",
  "like": "*api"
}, {
  "field": "Service.Web/sites/httpsOnly",
  "equals": "false"
} ] },
"then": { "effect": "Audit" }

Mistake ⇒
- No audit occurs for noncompliant resource
- Audit occurs for compliant resource
Anatomy of a Security Policy

API App should only be accessible over HTTPS

```
"if": { "allOf": [ 
  { "field": "type", 
    "equals": "Service.Web/sites" 
  }, 
  { "field": "kind", 
    "like": "*api" 
  }, 
  { "field": "Service.Web/sites/httpsOnly", 
    "equals": "false" 
  } ] },
"then": { 
  "effect": "Audit"
}
```

Boolean Combination
- "allOf" = AND
- "anyOf" = OR
- "not" = NOT

Regex constraints
matches(kind, .*api )

Scope of Policy

12
Automated Security Verification

• Date format validation:
  "date" matches ####-??-##
  "date" matches 2019* or 2020*
  "date" matches *30 or *31 or *2? or *1? or *0?
  "date" does not match *00
  "date" contains "jan" or "feb" or …
  ...

\[ date \in \{d\{4\}-[a-zA-Z]\{3\}-d\{2\} \land (date \in 2019.\ast \lor date \in 2020.\ast) \land \ldots \} \]
State-of-the-Art Solvers on Date Format Example

INFO:root:Results at ./. benchmarks/temp/date_format_ days_months_complete_sat.smt2:
  z3: Timeout (10s)
  z3str3: Wrong
  z3trau: Timeout (10s)
  cvc4: Timeout (10s)
Diagnosis: Solvers Based on Automata

all of:
"date" matches 0* or 1* or 2* or 30*
"date" doesn’t match 00*
"date" contains "jan" or "feb" or …
...

Regex → state machine
Is final state reachable?
Diagnosis: Solvers Based on Automata

Boolean combination:
Union, intersection, complement

all of:
"date" matches 0* or 1* or 2* or 30*
"date" doesn’t match 00*
"date" contains "jan" or "feb" or ...

Regex → state machine
Is final state reachable?
Partial Solution from Formal Language Theory: "Derivatives"

\[ \partial_x(L) := \{ w : xw \in L \} \]
Partial Solution from Formal Language Theory: "Derivatives"

\[
\partial_x(L) := \{w : xw \in L\}
\]

\[
\partial_j \{\text{june, july, august} \} = \{\text{une, uly}\}
\]

\[
\partial_a \{\text{june, july, august} \} = \{\text{ugust}\}
\]

\[
\partial_b \{\text{june, july, august} \} = \{\}
\]
Derivatives of Regular Expressions

\[ \partial_x (L) := \{ w : xw \in L \} \]

\[ \partial_j (\text{june} \cup \text{july}) = \text{une} \cup \text{uly} \]

\[ \partial_j ((\text{june}^*) ) = \text{une}(\text{june})^* \]

\[ \partial_j (\text{.*june.*}) = (\text{.*june.*}) \cup (\text{une.*}) \]
Derivatives of Regular Expressions

\[ \partial_x (L) := \{ w : xw \in L \} \]

\[ \partial_j (\text{june} \cup \text{july}) = \text{une} \cup \text{uly} \]

\[ \partial_j ((\text{june})^*) = \text{une}(\text{june})^* \]

\[ \partial_j (.*\text{june}.* ) = (.*\text{june}.* ) \cup (\text{une}.* ) \]

Simple Algebraic Laws: \[ \partial_x (R^*) = \partial_x (R) R^* \]

=> purely functional pattern matching
Derivatives in SMT

\[ s \in (.*\text{july}.*\text{)} \text{satisfiable?} \]

\[ \text{hd}(s) = x \text{ and } \text{tl}(s) \in \partial_x (.*\text{july}.*\text{)} \text{satisfiable?} \]

\[ \text{hd}(s) = x \text{ and } \begin{cases} x = j \text{ and } \text{tl}(s) \in (\text{uly}\text{.*)} & \text{satisfiable?} \\ \text{or } \text{tl}(s) \in (.*\text{july}.*\text{)} & \end{cases} \]

"Lazy" strategy

Treat regex as a black box: try out one character at a time
Have We Solved the Problem?

Boolean combination:
Union, intersection, complement

all of:
"date" matches 0* or 1* or 2* or 30*
"date" doesn’t match 00*
"date" contains "jan" or "feb" or ...
...

\[ s \in \ldots (\{\text{.*} \text{jan} \text{.}*\} \cup \{\text{.*} \text{feb} \text{.}*\} \cup \ldots) \]

... satisfiable?
Additional Limitation: Large Character Space

\[ \partial_x(L) := \{ w : xw \in L \} \]

Need to know x to compute!

For SMT satisfiability: keep x as a symbol "symbolic setting"

Traditional solution: exhaustively list all finitely many possibilities

\[
\begin{cases}
    x = j \text{ and } \text{tl}(s) \in (uly.*) \\
    \text{or } \text{tl}(s) \in (. * july.*)
\end{cases}
\]

1. Possibly exponential enumeration up front
2. Blow up in regex complement!
Solution: Augment Regexes with `if-then-else`

\[ \partial_x (\text{june} \cup \text{july}) = \text{if } x = j \text{ then } \text{une} \cup \text{uly} \text{ else } \bot \]

**Key technical insight**

- Enables lazy solver constructions to go through
- Avoids exponential enumeration up front
- Naturally handles intersection and complement
Derivatives: Intersection and Complement

\[
\partial_x (\text{june} \cap \text{july}) = \partial_x (\text{june}) \cap \partial_x (\text{july})
\]

\[
\partial_x (\text{feb}^C) = (\partial_x (\text{feb}))^C
\]

\[
(\text{if } x = j \text{ then } \text{une} \cup \text{uly} \text{ else } \bot)^C
\]

\[
= \text{if } x = j \text{ then } (\text{une} \cup \text{uly})^C \text{ else } (\bot)^C
\]
Overall Decision Procedure

- Incremental graph reachability

- States = seen regexes (1, 2, 3, 4)

- States can be:
  - Live
  - Dead
  - Unknown
  - Not fully explored
Back to Date Format Example

INFO:root:Results at ../../../benchmarks/temp/date_format_days_months_complete_sat.smt2:
  dz3.v4: sat (2.02s)
  z3: Timeout (10s)
  z3str3: Wrong
  z3trau: Timeout (10s)
  cvc4: Timeout (10s)
Experiments

• Does dZ3 match the state of the art on standard benchmarks?

• Division of benchmarks:
  • Standard
  • Boolean
  • Handwritten
Results
Results: "Boolean" Benchmarks
dZ3: A New Regex Constraint Solver

Theory: Symbolic Boolean Derivatives of Regular Expressions

Try it out!
github.com/Z3Prover/z3

Benchmarks:
github.com/cdstanford/regex-smt-benchmarks

Experimental scripts:
github.com/cdstanford/dz3-artifact