ORACLE

Synthesis of Java Deserialisation Filters from Examples

Kostyantyn Vorobyov, Francois Gauthier, Sora Bae, Padmanabhan Krishnan and Rebecca O'Donoghue

Oracle Labs, Australia

June, 2022

Deserialisation in Java

Serialisation/deserialisation

- Convert an object into a stream of bytes and back
- Natively supported by Java[1]

Deserialisation of untrusted data

- Carefully crafted payload can trigger arbitrary functionality
- Over 600 CVEs reported in the last 5 years

Beyond native Java serialisation

- Jackson-databind: JSON-based serialisation
  - 9th most popular package on Maven as of May 2022
- Over 60 CVEs reported since 2017

[1] Java is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Deserialisation Filtering

Production-time monitor

- Validates contents of deserialised objects

Relies on user-provided filters

- *Blocklists*: block deserialisation of unsafe classes (less safe)
- *Allowlists*: allow deserialisation of benign classes (more safe)

Available tools:

- *JEP 290* (JDK[1])
  - First appeared in Java 9, backported to Java 6, 7 and 8
- *contrast-rO0* (Contrast Security)
- `ValidatingObjectInputStream` (Commons Collection)

[1] JDK is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Deseralisation Filters

Typically implemented as regular expressions over class names

Manual construction and maintenance of deserialisation filters is tedious and error prone

- Especially for large systems comprising many components

Best delegated to an automated approach

- Synthesise filters (as regular expressions) from examples
- Block deserialisation of potentially dangerous classes
- Allow deserialisation of benign yet previously unseen classes

Synthesis of Regular Expressions from Examples

Existing techniques

- Automata-theoretic
- Genetic programming
- Multiple sequence alignment

Not well suited for synthesis of deserialisation filters

- Either too specific or overly generic
- High cost (esp. automata-theoretic)
- Synthesised regular expressions are difficult to maintain
  - Reason at the level of individual characters

Can we synthesise accurate and manually auditable deserialisation filters at low cost?

ds-prefix: Automatic Synthesis of Deserialisation Filters from Examples

Focus

- Synthesis of allowlists (regular expressions) from benign and malicious examples (class names)
  - An example matching the generated allowlist should be allowed and blocked otherwise

Observation

- Existing filters often reason at the level of packages rather than individual classes
  - Allow or block deserialisation of classes with given prefixes

Key ideas

- Find shortest prefixes that describe all positive examples and none of the negative
- Generalise concrete class names

Positive, Negative and Conflicting Prefixes

Examples

- $S_+ = \{java.lang.Byte, java.lang.Short\}$
- $S_- = \{java.io.Writer\}$

Prefixes

- $java$: conflicting
- $java.lang$: positive
- $java.io$: negative

Regular expression

- Accept any class starting with a positive prefix
  - `java\.lang\..*`

Resolving Conflicting Prefixes

Examples

- $S_+ = \{java.lang.String\}$
- $S_- = \{java.lang.Runtime\}$

Additive approach

- Accept only positive examples
  - `java\.lang\.String`

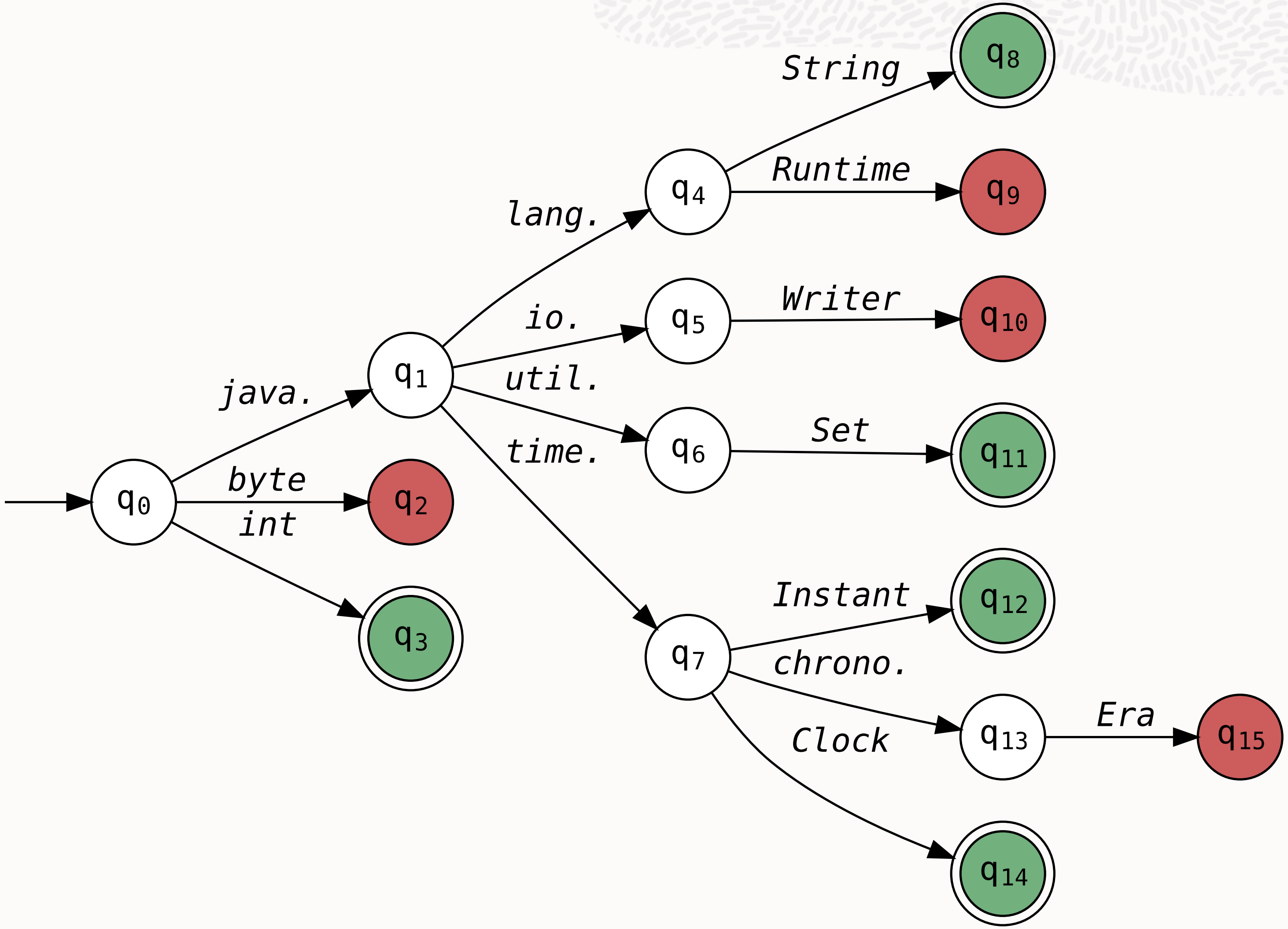
Subtraction approach

- Accept any example from the same package except negative
  - `java\.lang\.(?!Runtime$)[^.]+`

Augmented Prefix Tree Acceptor (APTA) over Java Class Names

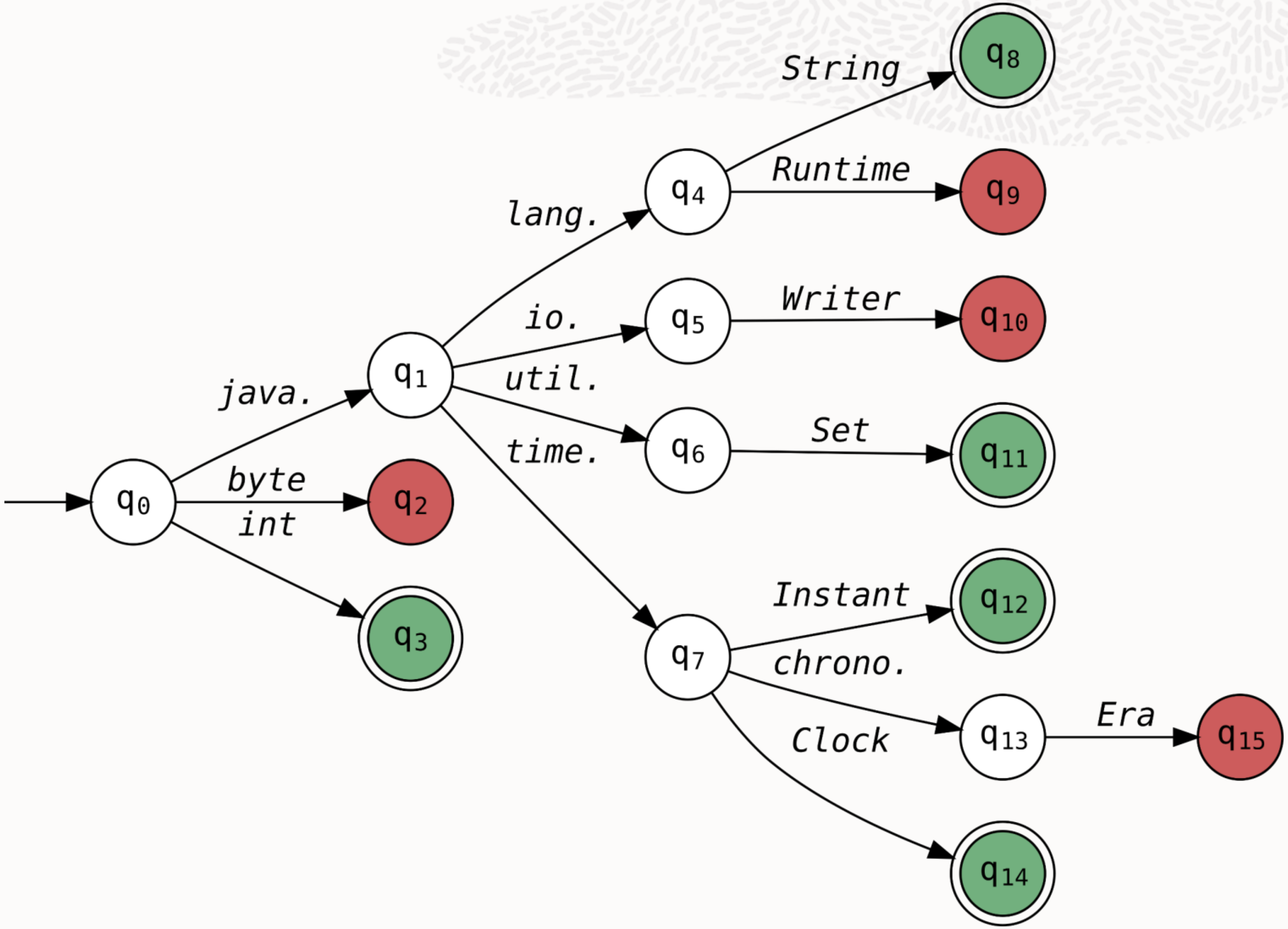
S_+
int
java.lang.String
java.util.Set
java.time.Instant
java.time.Clock

S_-
byte
java.lang.Runtime
java.io.Writer
java.time.chrono.Era



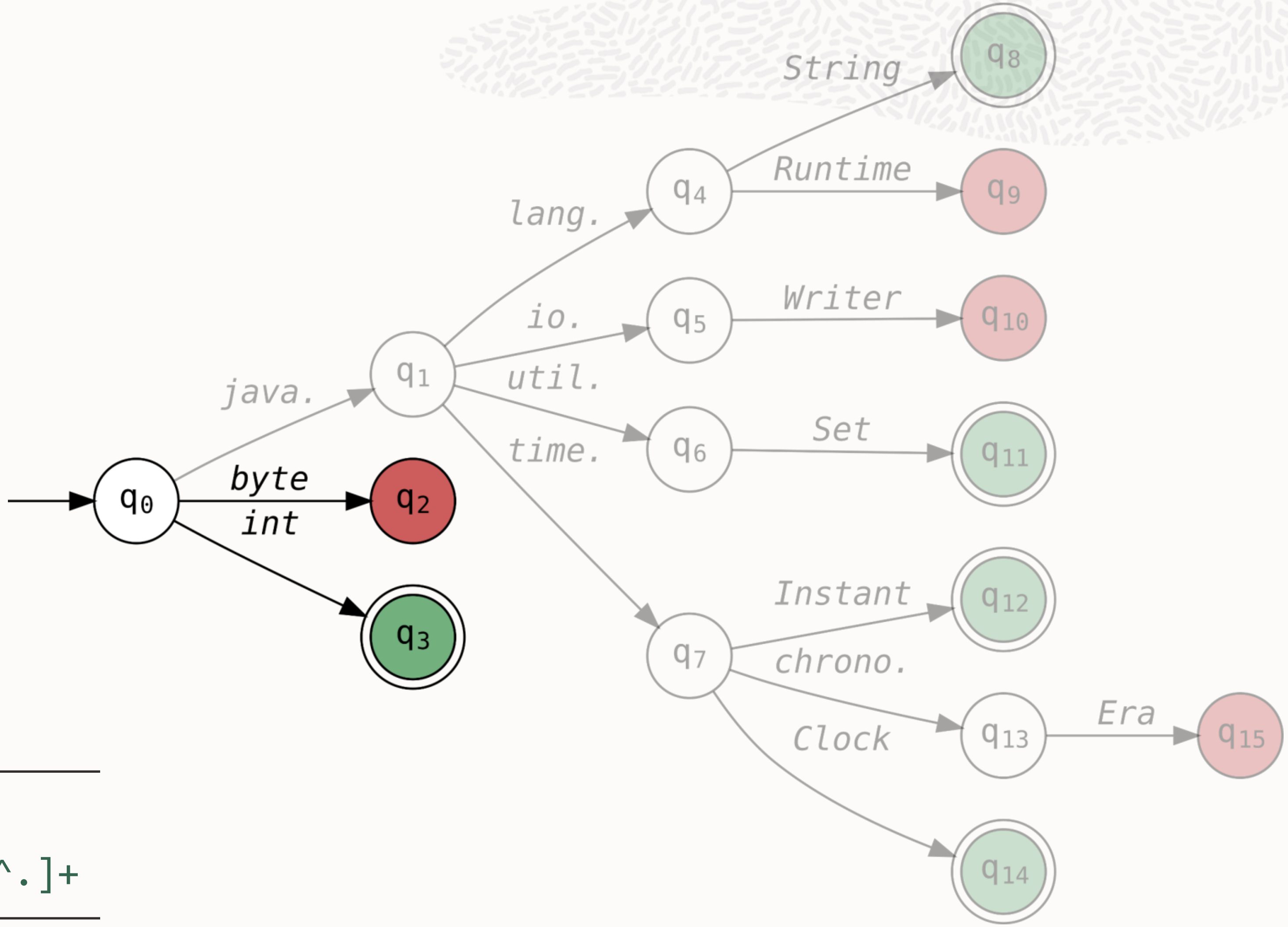
Synthesis Example

Current state: q_0



Synthesis Example

Current state: q_0



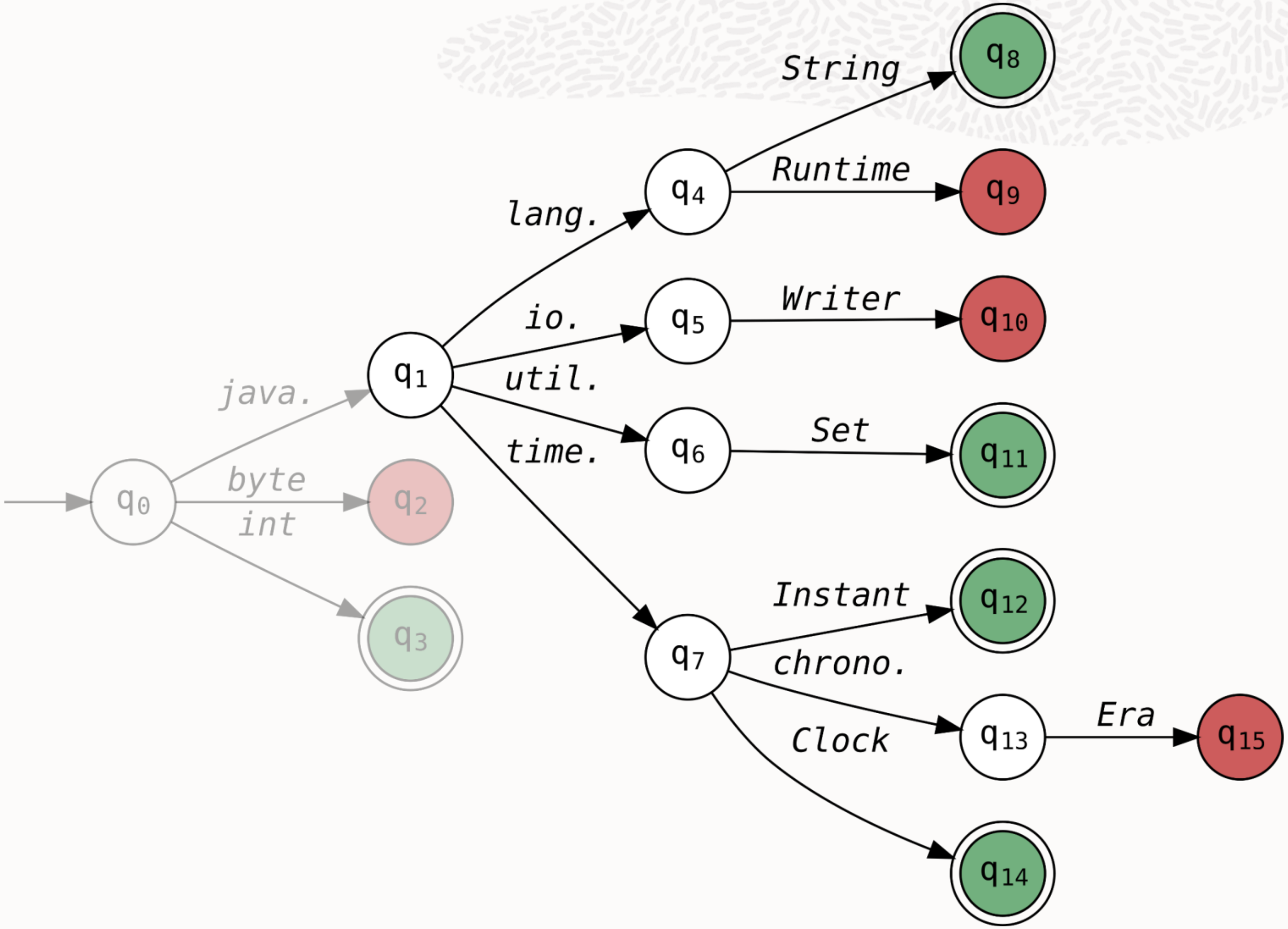
Regex (additive): `int`

Regex (subtraction): `(?!byte$)[^.]+`

June, 2022

Synthesis Example

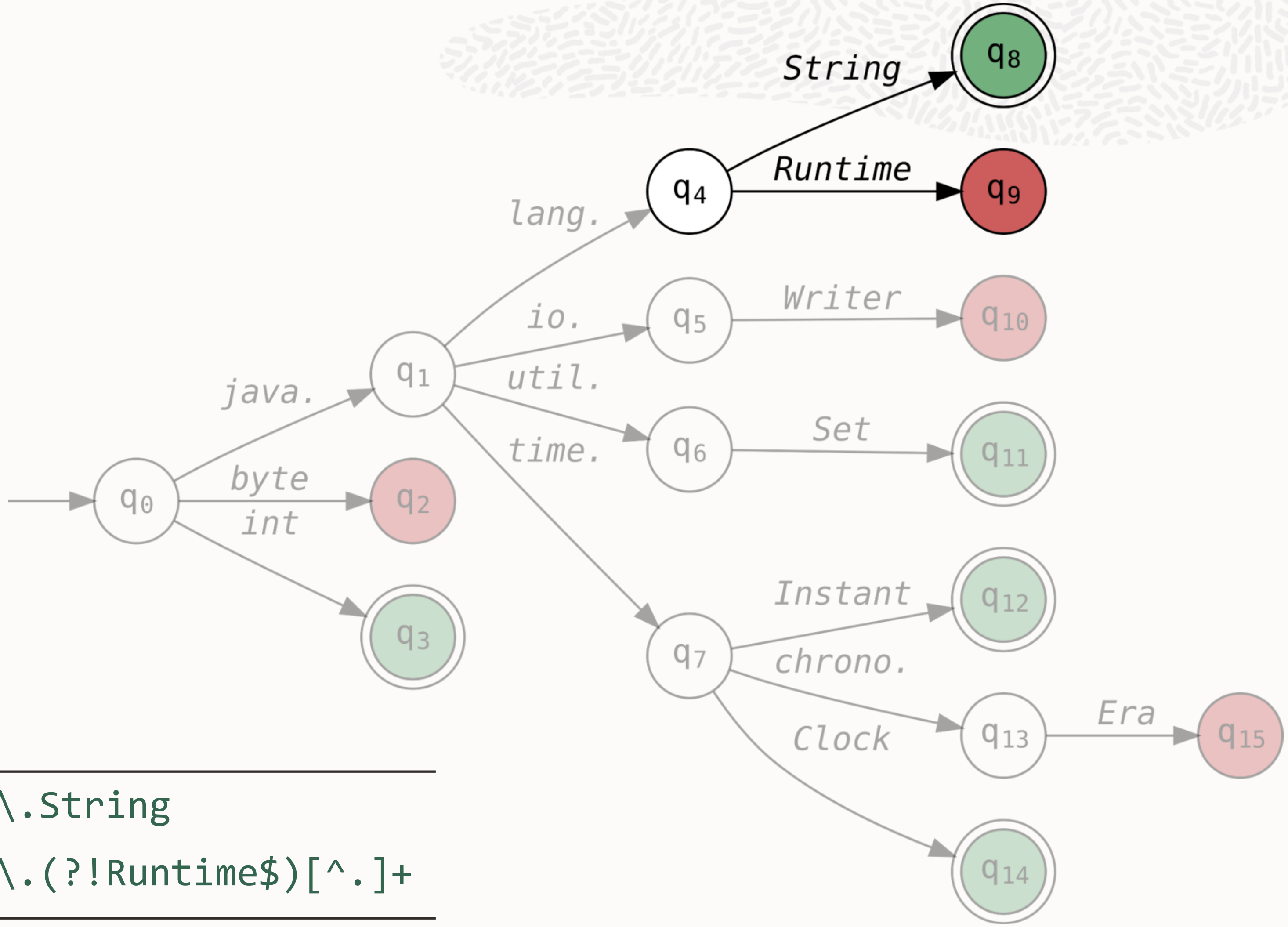
Current state: q_1



  June, 2022

Synthesis Example

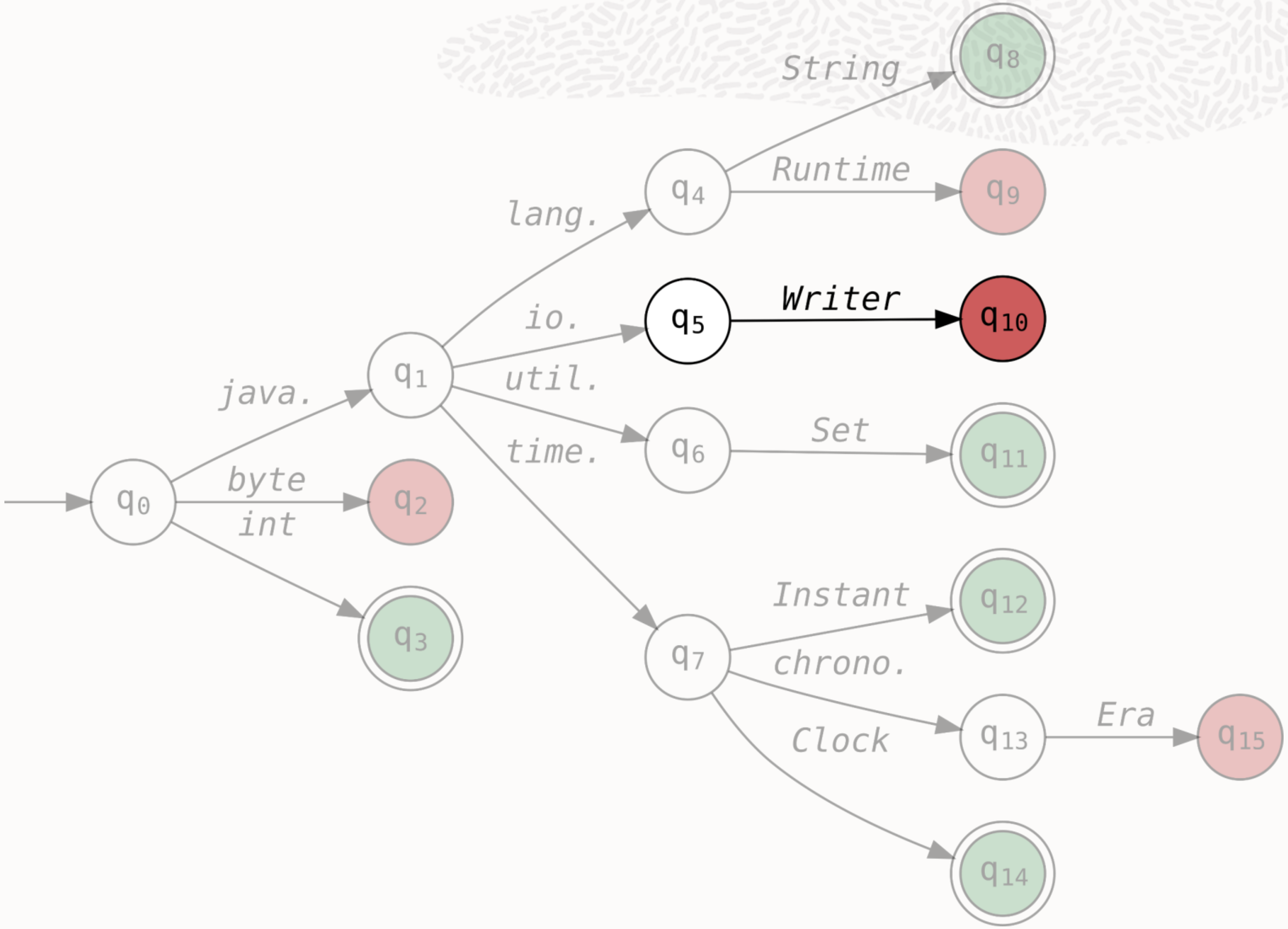
Current state: q_4



Regex (additive):	`java\.lang\.String`
Regex (subtraction):	`java\.lang\.(?!Runtime$)[^.]+`

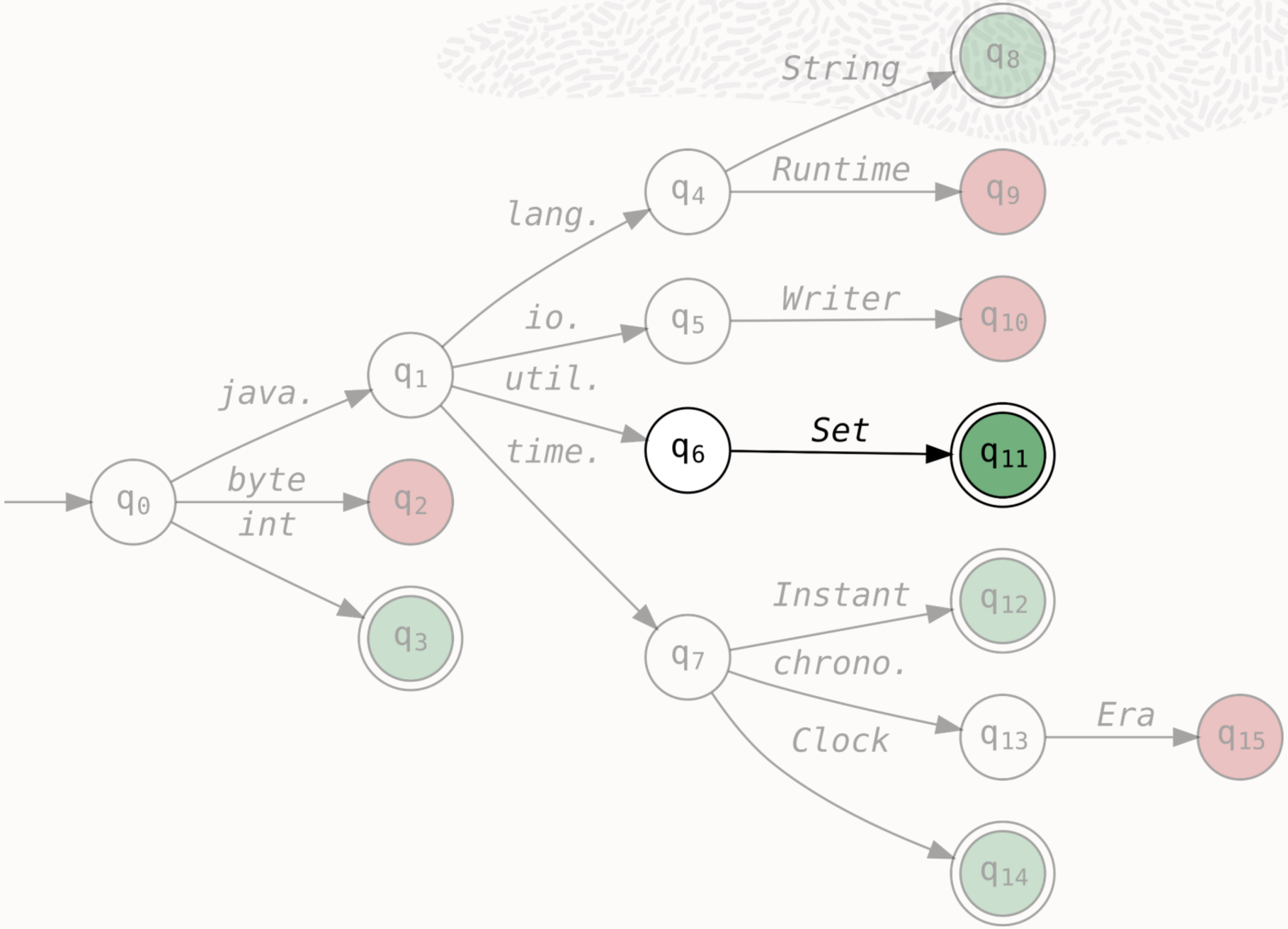
Synthesis Example

Current state: q_5



Synthesis Example

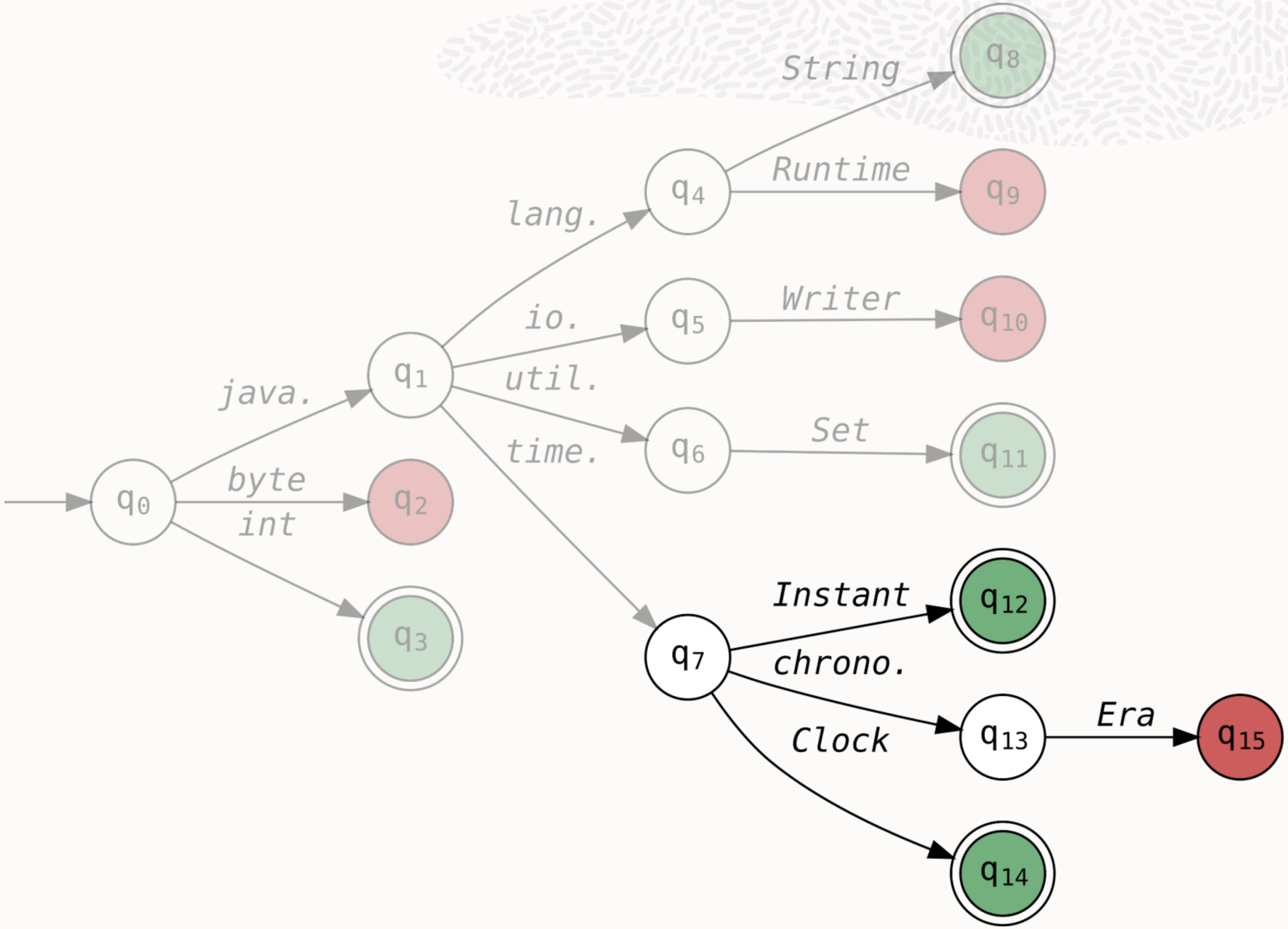
Current state: q_6



Regex: `java\.util\..*`

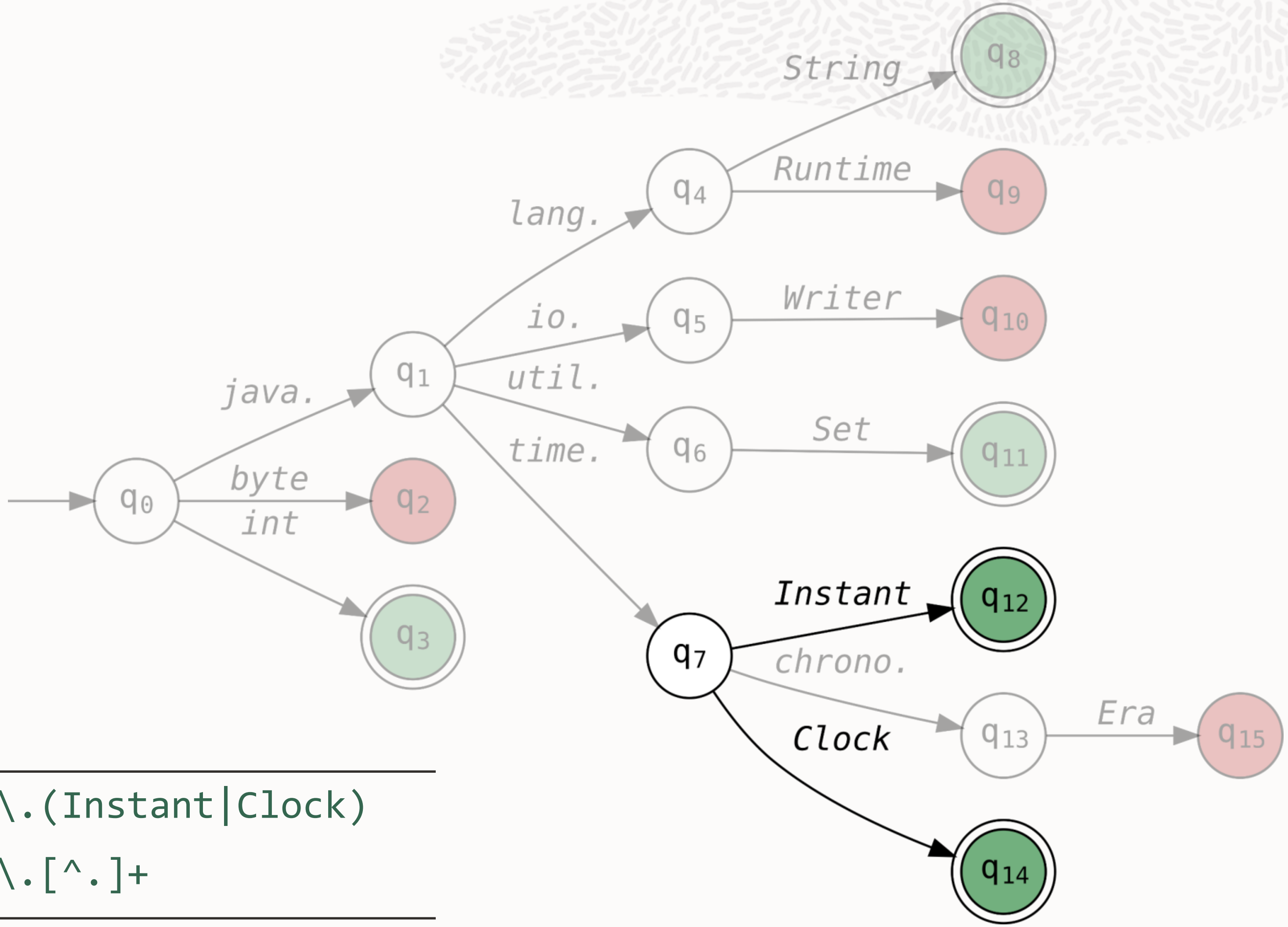
Synthesis Example

Current state: q_7



Synthesis Example

Current state: q_7

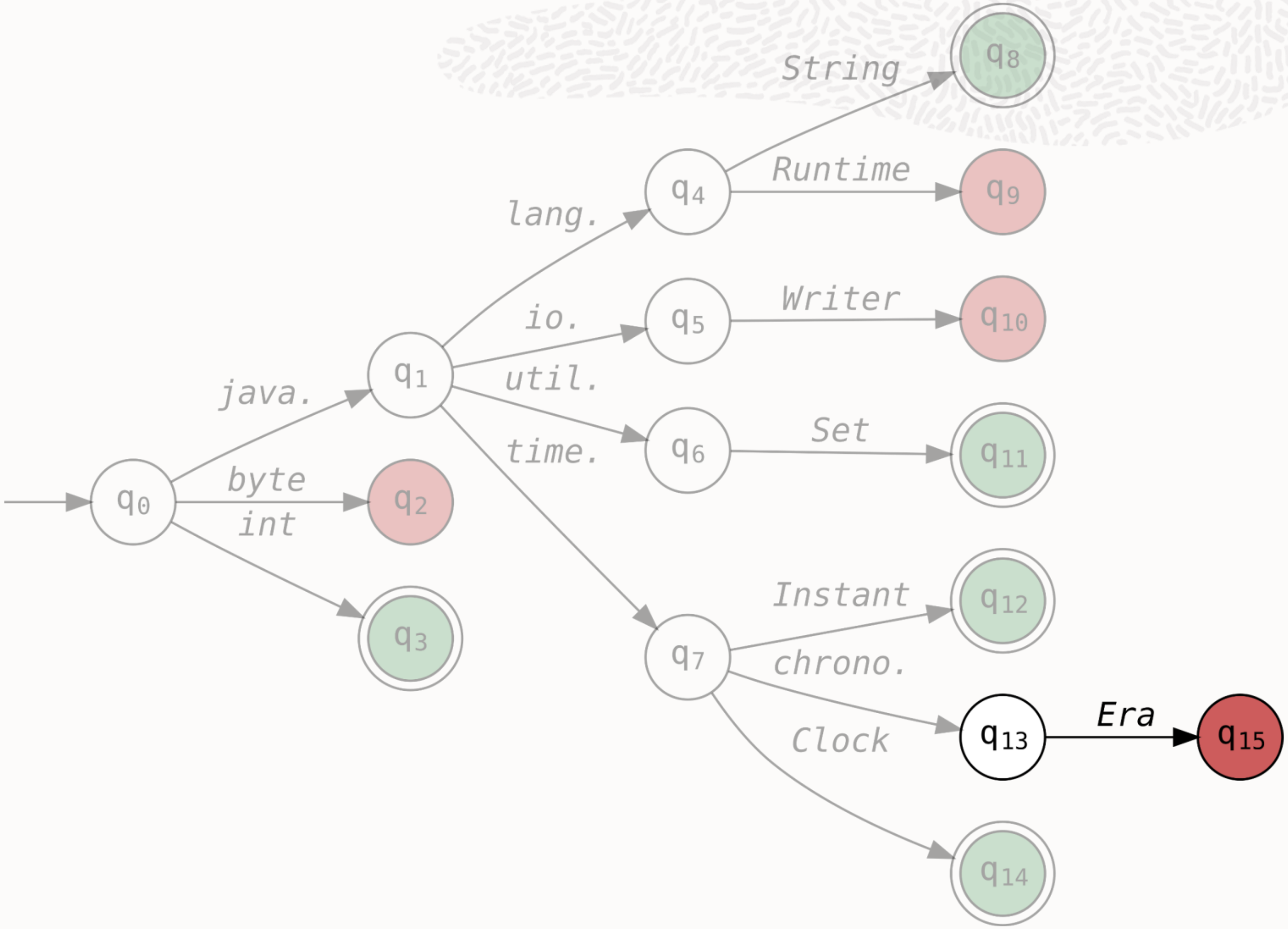


Regex (additive): `java\.time\.(Instant|Clock)`

Regex (subtraction): `java\.time\.[^.]+`

Synthesis Example

Current state: q_{13}



Synthesis Example

S_+	S_-
int	$byte$
$java.lang.String$	$java.lang.Runtime$
$java.util.Set$	$java.io.Writer$
$java.time.Instant$	$java.time.chrono.Era$
$java.time.Clock$	

Regex (additive):

```
^int|java\.lang\.String|java\.time\.(Instant|Clock)|java\.util\..*$
```

Regex (subtraction):

```
^(?!byte$)|java\.lang\.(?!Runtime$)[^.]+|java\.time\.[^.]+|java\.util\..*$
```


Evaluation

ds-prefix synthesis

- Implemented using *dk.brics.automaton* library

Monitoring agent

- Collect names of deserialised classes (logging mode)
- Enforce specified allowlist (blocking mode)
- Allows deserialisation filtering in JDK (JEP 290) and Jackson-databind

Experiments

- Investigate applicability of *ds-prefix* to real deserialisation vulnerabilities
- Investigate precision and performance of *ds-prefix*
  - *Compare to state-of-the-art synthesis tools*

Vulnerability Detection

Experiment with vulnerable open-source projects

- Can *ds-prefix* allowlists prevent real vulnerabilities?

Methodology

- Reproduce a known vulnerability
- Gather examples and synthesise the allowlist
  - Positive examples gathered from test runs
  - Negative examples collected from application's blocklist and known gadget chains
- Confirm that the generated allowlist prevents the exploit

Vulnerability Detection

Name	Versions	CVE	Synthesised Allowlist
Olingo	4.0.0-4.7.0	CVE-2019-17556	`^org\.apache\.olingo\..+$`
Apache Batik		CVE-2018-8013	`^\[Lorg|com\.sun\.org\.apache\.xerces|com\.sun\.org\.apache\.xml|org\.apache\.batik|org\.apache\.html|org\.apache\.wml|org\.apache\.xerces|org\.apache\.xml|org\.python|org\.w3c\..+$`
Jackson-databind	2.9.x	CVE-2017-17485	`^((\[Lcom|\[Ljava|com\.fasterxml|java\.io|java\.lang|java\.text|java\.util\.concurrent)\..+|[^.]+|java\.util\.[^.]+)$`

Jackson-databind: Historic datasets

Datasets

- Datasets drawn from the blocklist of Jackson-databind after discovery of each CVE
- Initial dataset (9 negative examples, 1 known CVE)
  - Allowlist: `^((\[Lcom|\[Ljava|com\.fasterxml|java) \..+|[^.]+)$`
- Latest dataset (134 negative examples, 46 known CVEs)

Results

- Initial allowlist blocks 132 malicious classes (prevents 44 CVEs)
- Allowlist based on negative examples after discovery of the 4th CVE (48 examples) is sufficient to prevent deserialisation of known malicious classes

Comparison with Regular Expression Synthesis Tools

Automata-theoretic algorithms

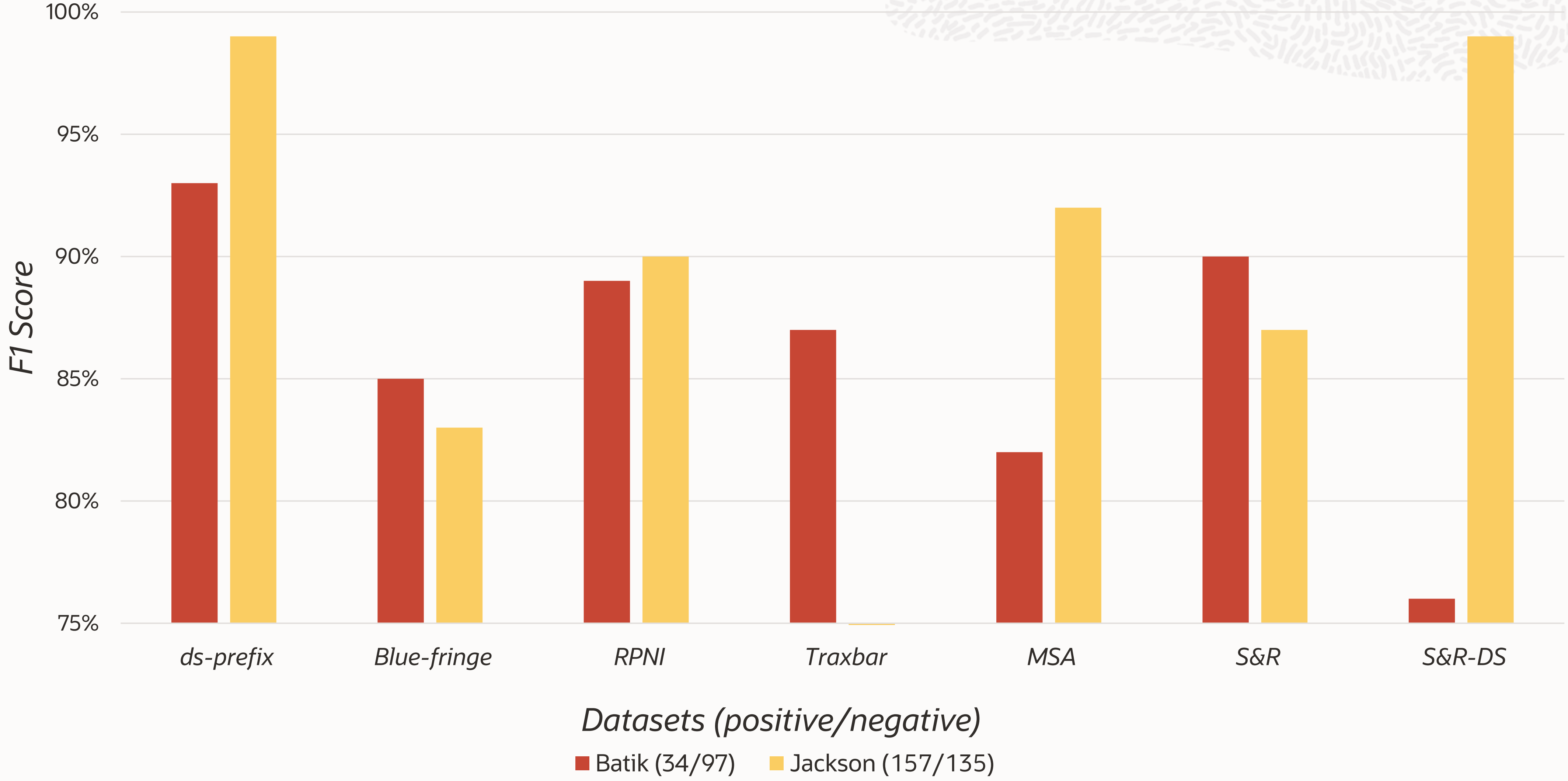
- Regular Positive Negative Inference *(RPNI)*
- Trakhtenbrot and Barzdin (*Traxbar*)
- *Blue-fringe*

Genetic programming

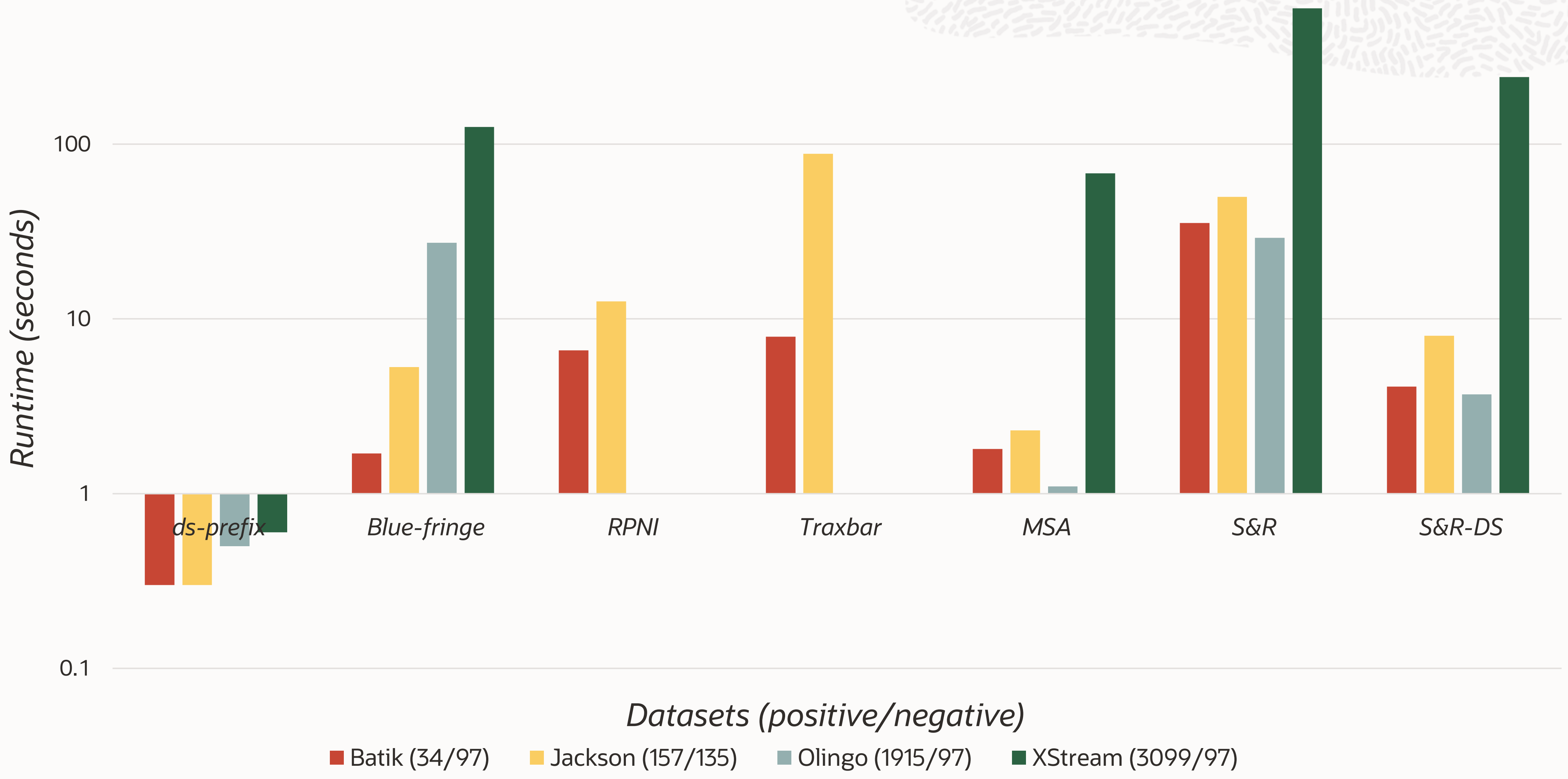
- Search and Replace Generator with character alphabet (*S&R*)
- Search and Replace Generator with alphabet of Java sub-packages and class names (*S&R-DS*)

Multiple Sequence Alignment (*MSA*)

F1 Score (5-fold Cross Validation)



Runtime Performance



  June, 2022

Auditability of Results

ds-prefix vs automata-theoretic algorithms

ds-prefix

```
^(\[Lorg|com\.sun\.org\.apache\.xerces|com\.sun\.org\.apache\.xml|org\.apache\.batik|org\.apache
\.html|org\.apache\.wml|org\.apache\.xerces|org\.apache\.xml|org\.python|org\.w3c)\..+$
```

Blue-fringe

```
^([a-zA-CE-HJLMOPR-X02-46\$\.\[]|[DIN;]([cmopS]|([dtu]|l[enp])([iI]|[mo][enp])*[aelnC])*([ aenrB-
DMOPRT]|([dtu]|l[enp])([iI]|[mo][enp])*E))*[DIN;]([cmopS]|([dtu]|l[enp])([iI]|[mo][enp])*[aelnC])*((
[dtu]|l[enp])([iI]|[mo][enp])*)?$
```

RPNI

```
^([a-ce-ik-mopr-uw-yAC-EG-IL-PTV-X3\$\.\[]|[dnS][del-nptIS]*[a-cf-ikorsuw-yAC-EGHL-PTV-X3\$\.\[])*([
dnS][del-nptIS]*|([dnS][deptIS]*)?;[elmptI]*)$
```

Auditability of Results

ds-prefix vs genetic programming

ds-prefix

```
^(\[Lorg|com\.sun\.org\.apache\.xerces|com\.sun\.org\.apache\.xml|org\.apache\.batik|org\.apache
\.html|org\.apache\.wml|org\.apache\.xerces|org\.apache\.xml|org\.python|org\.w3c)\..+$
```

S&R

```
^[^l]++[^p]++(?:[^m]++[^r]++)++$
```

S&R-DS

```
:ˆ(\[L?\.?|org\.?|xerces\.?)([ˆ.]+\.?)(xml\.?|html\.?|wml\.?|org\.?|apache\.?|batik\.?|dom\.?|xerces
\.?)([^.]+\.?)++;?$
```


Auditability of Results

ds-prefix vs multiple sequence alignment

ds-prefix

```
^(\[Lorg|com\.sun\.org\.apache\.xerces|com\.sun\.org\.apache\.xml|org\.apache\.batik|org\.apache
\.html|org\.apache\.wml|org\.apache\.xerces|org\.apache\.xml|org\.python|org\.w3c)\..+$
```

MSA

```
^\[Lorg.apache.batik.dom.AbstractElement\$Entry;$|^com.sun.org.apache.x.{2,5}.internal.{0,8}..{6,25}
Implementation.{0,4}$|^org.apache.{10,44}ent.{0,9}$|^org.python.apache.{0,27}DOMImplementation.{0,4}
$|^ org.w3c.dom.{0,5}..{4,4}DOMImplementation.{0,3}$
```

Conclusions

ds-prefix

- Synthesis of regular expressions that specifically targets deserialisation filtering
  - Find a set of shortest prefixes that describe all positive examples but none of the negative
- Reason at the level of packages and class names rather than individual characters
- Avoid costly conversion from finite automata to regular expressions

Well-suited for deserialisation filtering

- Prevents real exploits using a limited number of input examples
- Has the potential to block future attacks
- More precise and considerably faster then other synthesisers
- Produces manually auditable regular expressions

Thank you

kostyantyn.x.vorobyov@oracle.com
francois.gauthier@oracle.com
sora.bae@oracle.com
paddy.krishnan@oracle.com
rebecca.o.donoghue@oracle.com

https://labs.oracle.com/

