Validating Streaming JSON Documents with Learned Visibly Pushdown Automata

Published at TACAS 2023

Véronique Bruyère, Guillermo A. Pérez, Gaëtan Staquet

Theoretical computer science
Computer Science Department
Science Faculty
University of Mons

Formal Techniques in Software Engineering
Computer Science Department
Science Faculty
University of Antwerp

April 6, 2024

UMONS Université de Mons

fnrs LA LIBERTÉ DE CHERCHER

University of Antwerp

```json
{
  "title": "Validating JSON documents",
  "place": {
    "town": "Luxembourg",
    "country": "Luxembourg"
  }
}
```

```
{
  "title": "Validating JSON documents",
  "place": {
    "town": "Luxembourg",
    "country": "Luxembourg"
  }
}
```

An object is an unordered collection of key-value pairs.

There are also arrays (ordered collections of values); we ignore them in this talk.

```
{
  "title": "Validating JSON documents",
  "place": {
    "town": "Luxembourg",
    "country": "Luxembourg"
  }
}
```

An object is an unordered collection of key-value pairs.

There are also arrays (ordered collections of values); we ignore them in this talk.

We want to verify that the document satisfies some constraints.

```
{
  "title": "Validating JSON documents",
  "place": {
    "town": "Luxembourg",
    "country": "Luxembourg"
  }
}
```

"title" $\mapsto$ string
"place" $\mapsto$ object such that
"town" $\mapsto$ string
"country" $\mapsto$ string

An object is an unordered collection of key-value pairs.

There are also arrays (ordered collections of values); we ignore them in this talk.

We want to verify that the document satisfies some constraints.

Classical validation algorithm:

1. Explore the document and the constraints in parallel;
2. If the current value does not match the sub-constraints, stop;
3. Otherwise, repeat recursively.

Classical validation algorithm:

1. Explore the document and the constraints in parallel;
2. If the current value does not match the sub-constraints, stop;
3. Otherwise, repeat recursively.

The constraints can have Boolean operations.
$\hookrightarrow$ The same value must be processed multiple times.

Assume we are in a streaming context.

$\hookrightarrow$ We receive the document one fragment at a time.

The classical algorithm must wait for the whole document.

Assume we are in a streaming context.
↪ We receive the document one fragment at a time.
The classical algorithm must wait for the whole document.

Our approach:

- Construct an automaton from the constraints.
  - What kind of automaton?
  - How to construct it?
  - There is an exponential number of permutations of the keys.
- Abstract the automaton to know which part of the automaton reads an object.
- Validation algorithm using the automaton and the abstraction.

Assume we are in a streaming context.
$\hookrightarrow$ We receive the document one fragment at a time.
The classical algorithm must wait for the whole document.

Our approach:

- Construct an automaton from the constraints.
  - What kind of automaton?
  - How to construct it?
  - There is an exponential number of permutations of the keys.
- Abstract the automaton to know which part of the automaton reads an object.
- Validation algorithm using the automaton and the abstraction.

We can process the document while receiving it!

"rec" $\mapsto$ object such that

the object is empty

or the object is recursively defined

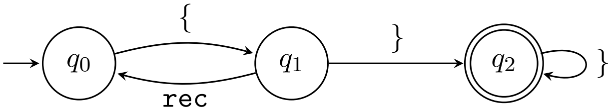


Figure 1: Recursive constraints and a deterministic finite automaton.

"rec" $\mapsto$ object such that

the object is empty

or the object is recursively defined

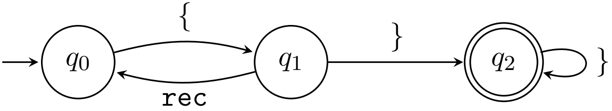


Figure 1: Recursive constraints and a deterministic finite automaton.

Accepts {rec{} ☹

"rec" $\mapsto$ object such that

the object is empty

or the object is recursively defined

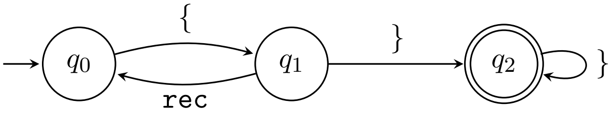


Figure 1: Recursive constraints and a deterministic finite automaton.

Accepts {rec{} ☹

The language $L = \{(\{\texttt{rec})^n\}^n \mid n > 0\}$ is not regular but can be described by our constraints.

Use a stack to remember how many objects we opened.

Use a stack to remember how many objects we opened.

$\hookrightarrow$ Visibly pushdown automata (VPA).

Use a stack to remember how many objects we opened.

↪ Visibly pushdown automata (VPA).

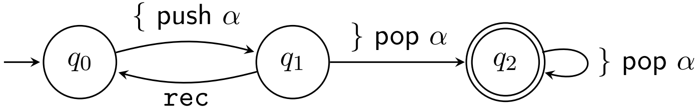


Figure 2: A VPA for the recursive constraints.

Theorem 1 (Contribution)

Let $\mathcal{C}$ be a set of constraints. Then, there is a VPA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A})$ is the set of documents valid with regards to $\mathcal{C}$.
If we fix an order $<$ over the keys, there is a VPA $\mathcal{B}$ accepting words that follow $<$. In some cases, $\mathcal{B}$ is exponentially smaller than $\mathcal{A}$.

Theorem 1 (Contribution)

Let $\mathcal{C}$ be a set of constraints. Then, there is a VPA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A})$ is the set of documents valid with regards to $\mathcal{C}$.
If we fix an order $<$ over the keys, there is a VPA $\mathcal{B}$ accepting words that follow $<$. In some cases, $\mathcal{B}$ is exponentially smaller than $\mathcal{A}$.

Theorem 2 (Isberner, "Foundations of active automata learning: an algorithmic perspective", 2015)

Let L be a language accepted by some VPA. Then, one can learn a VPA accepting L with a polynomial number of membership and equivalence queries.

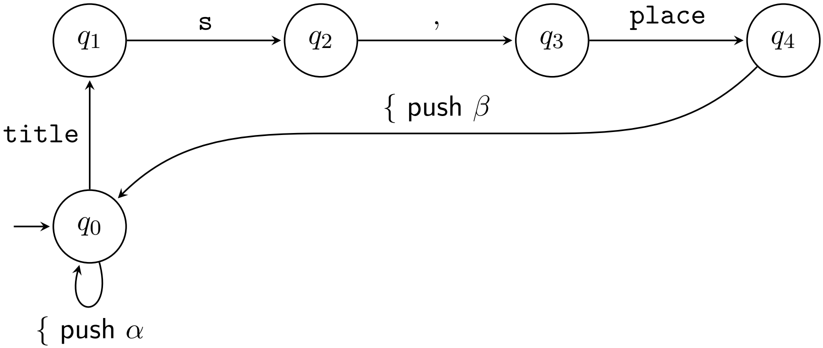
Theorem 1 (Contribution)

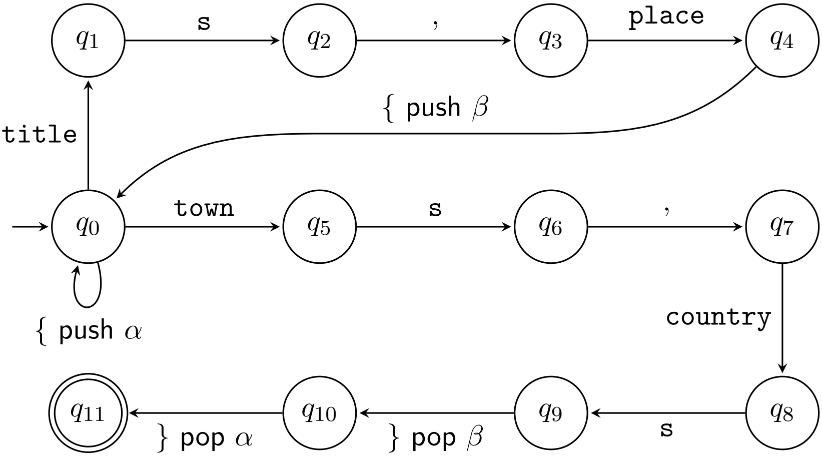
Let $\mathcal{C}$ be a set of constraints. Then, there is a VPA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A})$ is the set of documents valid with regards to $\mathcal{C}$.
If we fix an order $<$ over the keys, there is a VPA $\mathcal{B}$ accepting words that follow $<$. In some cases, $\mathcal{B}$ is exponentially smaller than $\mathcal{A}$.

Theorem 2 (Isberner, "Foundations of active automata learning: an algorithmic perspective", 2015)

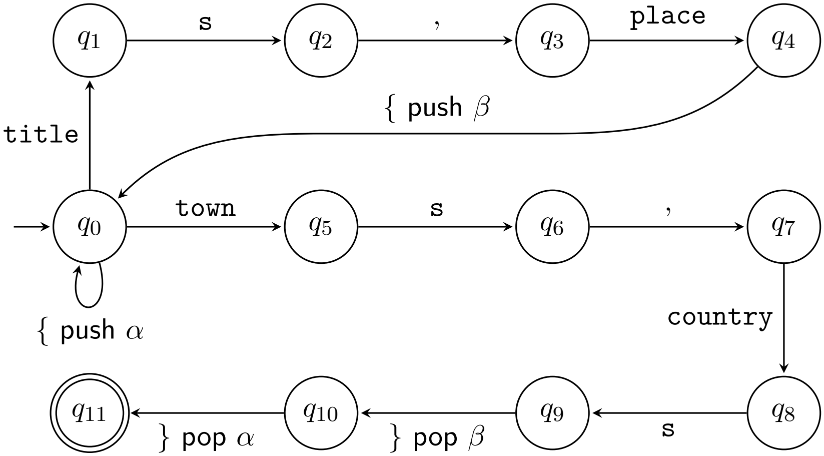
Let L be a language accepted by some VPA. Then, one can learn a VPA accepting L with a polynomial number of membership and equivalence queries.

$\hookrightarrow$ Given $\mathcal{C}$ and $<$, we learn a VPA.


q_1
s
q_2
,
q_3
place
q_4
{ push β
title
q_0
{ push α



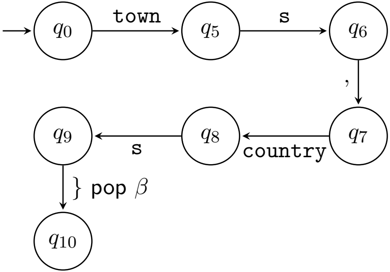
q_1
s
q_2
,
q_3
place
q_4
{ push β
title
q_0
town
q_5
s
q_6
,
q_7
{ push α
country
q_{11}
} pop α
q_{10}
} pop β
q_9
s
q_8



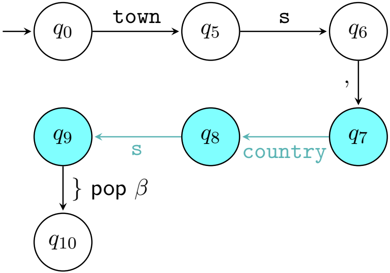
How can we read a document that does not follow the VPA's order?

$\{$ place $\{$ country s , town s $\}$, title s $\}$

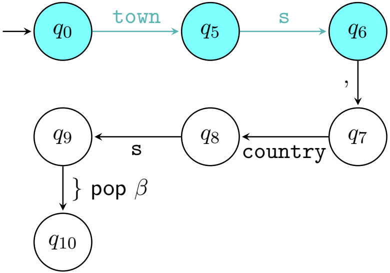
Let us focus on
{ country s , town s }.



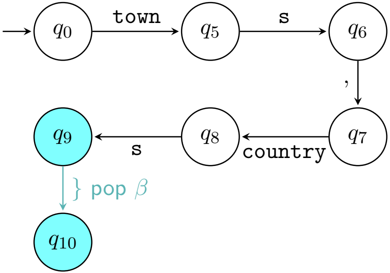
Let us focus on
{ country s , town s }.



Let us focus on
{ country s , town s }.

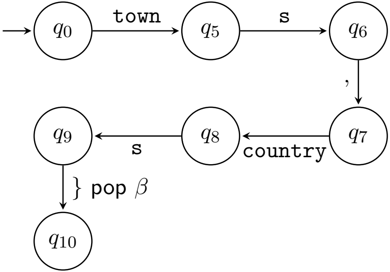


Let us focus on
{ `country` `s` , `town` `s` }.



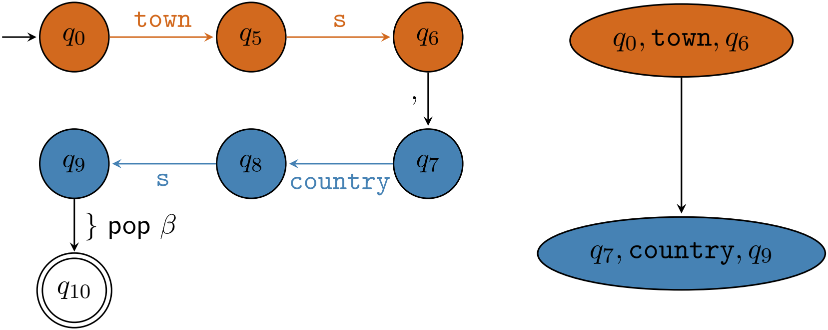
Let us focus on
$\{$ `country` `s` , `town` `s` $\}$.

$\hookrightarrow$ We need to "jump" around in the VPA.

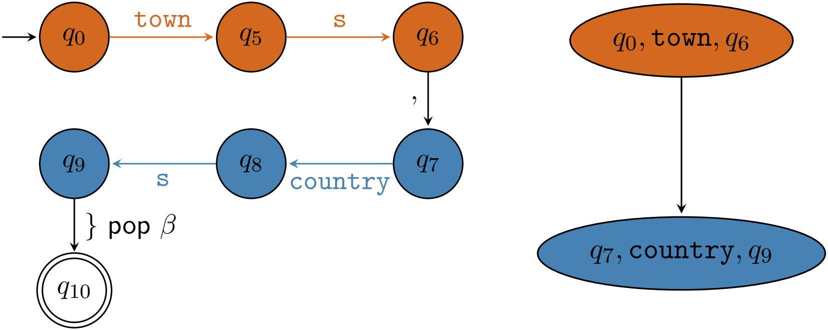


The key graph summarizes the possible jumps.

The key graph summarizes the possible jumps.

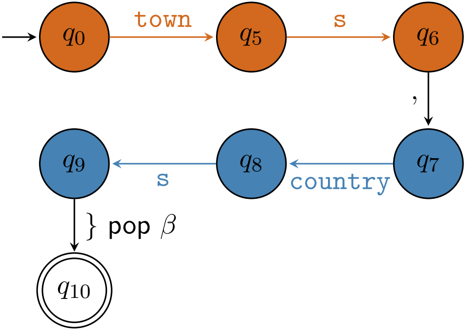


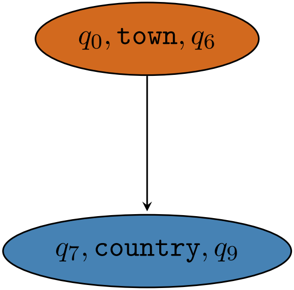
The key graph summarizes the possible jumps.



When we see the key `town`, we jump to q_0.
When we see the key `country`, we jump to q_7.

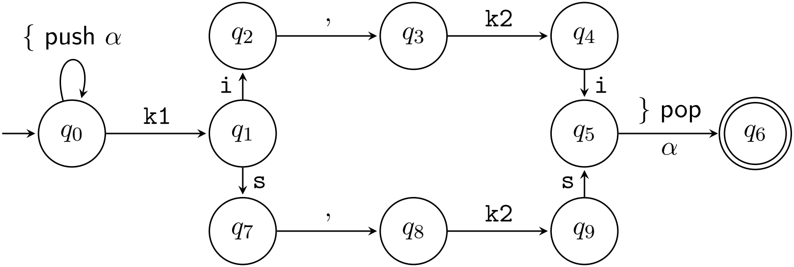
The key graph summarizes the possible jumps.

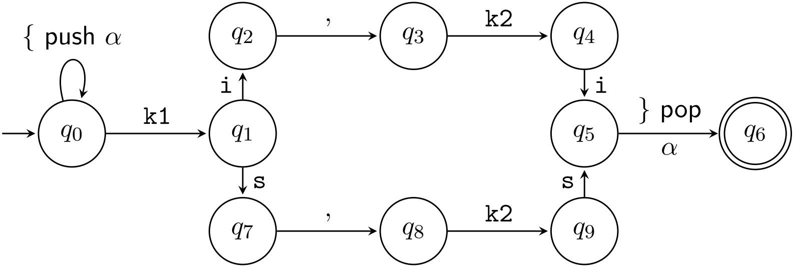


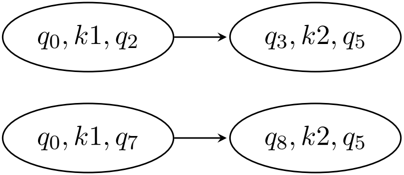


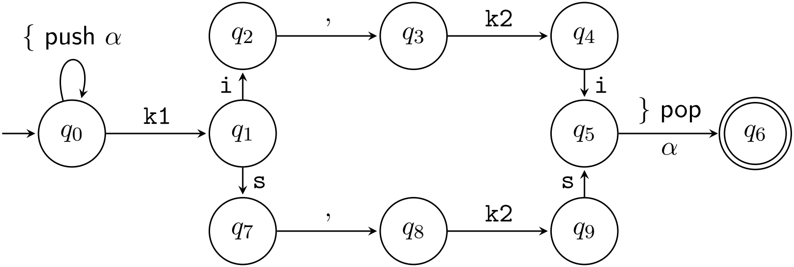
When we see the key `town`, we jump to q_0.
When we see the key `country`, we jump to q_7.

Is that enough?


{ push α
q_0
k1
q_1
i
q_2
,
q_3
k2
q_4
i
q_5
} pop
α
q_6
s
q_7
,
q_8
k2
q_9
s


{ push α
q_0
k1
q_1
i
q_2
,
q_3
k2
q_4
i
q_5
} pop
α
q_6
s
q_7
,
q_8
k2
q_9
s

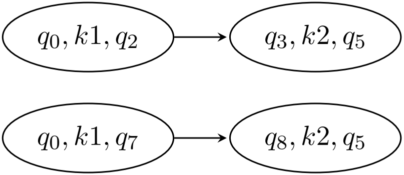

$q_0, k1, q_2$
$q_3, k2, q_5$
$q_0, k1, q_7$
$q_8, k2, q_5$

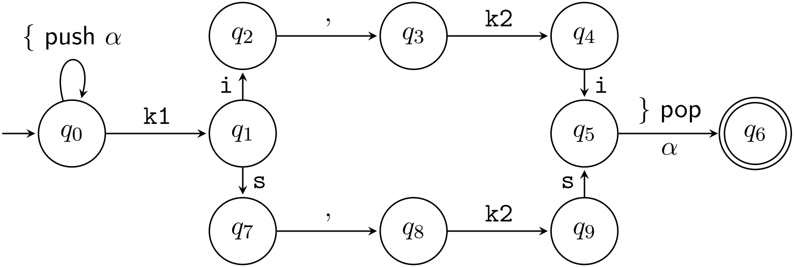


We need to distinguish between

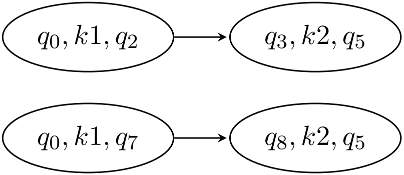
{ `k2 i , k1 i` }

{ `k2 s , k1 i` }.





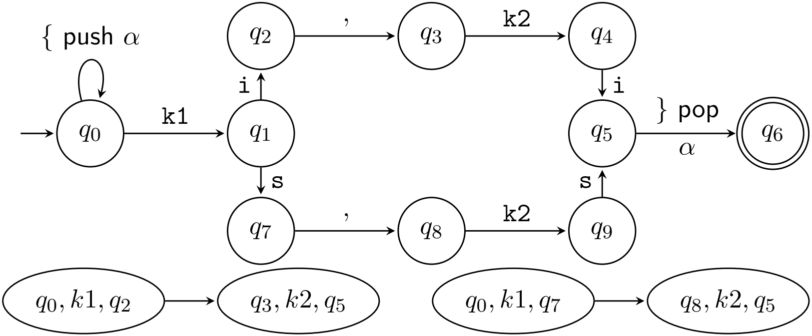
We need to distinguish between



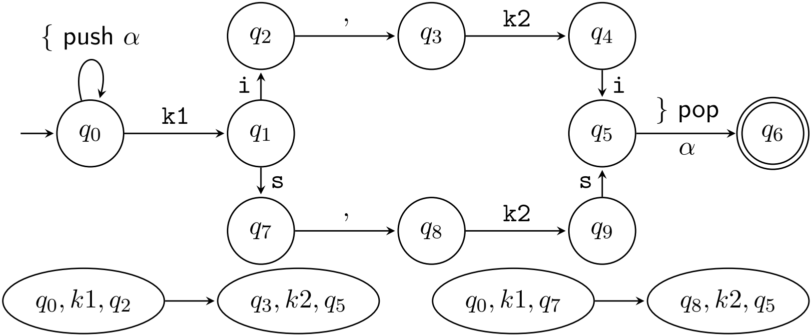
`{ k2 i , k1 i }`

`{ k2 s , k1 i }`.

$\hookrightarrow$ We must reconstruct the actual path.

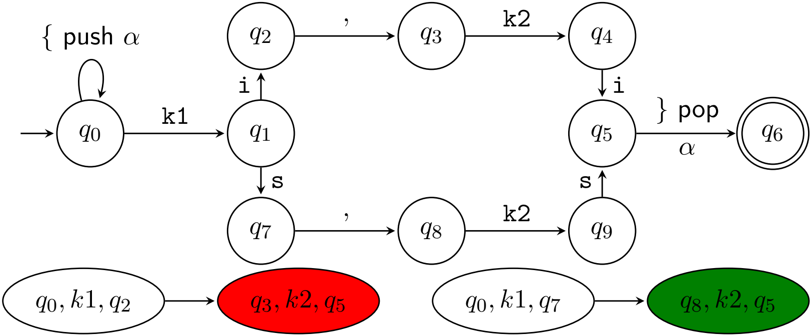


We read { `k2 s , k1 i` } $\rightsquigarrow$ set of pairs of states $\{(q_3, q_3), (q_8, q_8)\}$.



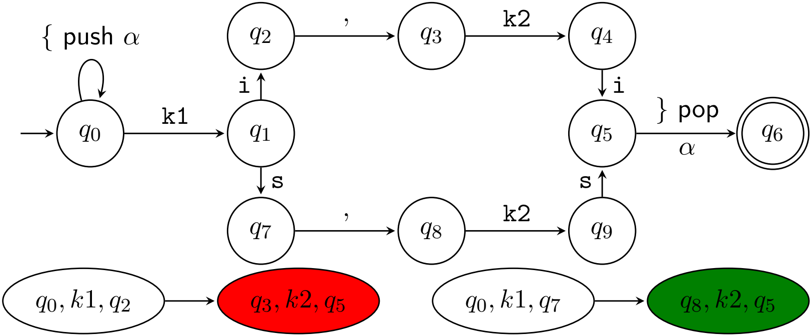
We read { k2 s , k1 i } ⇝ set of pairs of states $\{(q_3, q_3), (q_8, q_8)\}$.

1. Read k2 ⇝ $\{(q_3, q_4), (q_8, q_9)\}$.



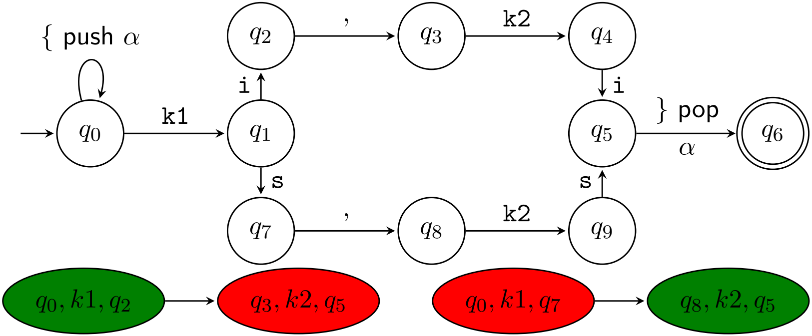
We read { `k2` `s` , `k1` `i` } $\rightsquigarrow$ set of pairs of states $\{(q_3, q_3), (q_8, q_8)\}$.

1. Read `k2` $\rightsquigarrow \{(q_3, q_4), (q_8, q_9)\}$.
2. Read `s` $\rightsquigarrow \{(q_8, q_5)\}$.



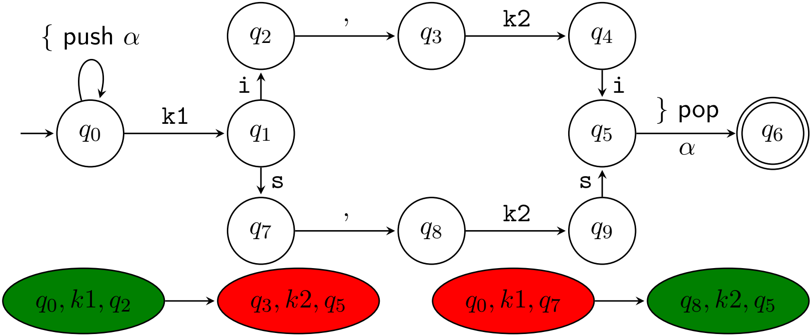
We read { k2 s , k1 i } $\rightsquigarrow$ set of pairs of states $\{(q_3, q_3), (q_8, q_8)\}$.

1. Read k2 $\rightsquigarrow$ $\{(q_3, q_4), (q_8, q_9)\}$.
2. Read s $\rightsquigarrow$ $\{(q_8, q_5)\}$.
3. Mark that $q_3 \to q_5$ was not seen.



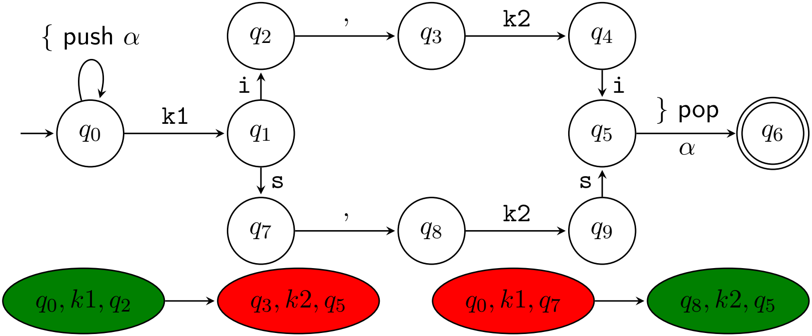
We read { `k2` `s` , `k1` `i` } $\rightsquigarrow$ set of pairs of states $\{(q_3, q_3), (q_8, q_8)\}$.

1. Read `k2` $\rightsquigarrow \{(q_3, q_4), (q_8, q_9)\}$.
2. Read `s` $\rightsquigarrow \{(q_8, q_5)\}$.
3. Mark that $q_3 \rightarrow q_5$ was not seen.
4. Read `k1` `i` and mark $q_0 \rightarrow q_7$ as not seen.



We read { `k2` `s` , `k1` `i` } $\rightsquigarrow$ set of pairs of states $\{(q_3, q_3), (q_8, q_8)\}$.

1. Read `k2` $\rightsquigarrow$ $\{(q_3, q_4), (q_8, q_9)\}$.
2. Read `s` $\rightsquigarrow$ $\{(q_8, q_5)\}$.
3. Mark that $q_3 \to q_5$ was not seen.
4. Read `k1` `i` and mark $q_0 \to q_7$ as not seen.
5. Did not see one of the possibilities.



We read { k2 s , k1 i } $\rightsquigarrow$ set of pairs of states $\{(q_3, q_3), (q_8, q_8)\}$.

1. Read k2 $\rightsquigarrow$ $\{(q_3, q_4), (q_8, q_9)\}$.
2. Read s $\rightsquigarrow$ $\{(q_8, q_5)\}$.
3. Mark that $q_3 \rightarrow q_5$ was not seen.
4. Read k1 i and mark $q_0 \rightarrow q_7$ as not seen.
5. Did not see one of the possibilities. $\rightsquigarrow$ We reject the word.

Ideas:

- From a VPA $\mathcal{A}$, construct its key graph.

Ideas:

- From a VPA $\mathcal{A}$, construct its key graph.
- Use the key graph to know where to “jump”.
  - Potentially, we are in multiple states at the same time (non-determinism).

Ideas:

- From a VPA $\mathcal{A}$, construct its key graph.
- Use the key graph to know where to "jump".
  - Potentially, we are in multiple states at the same time (non-determinism).
- During execution, use a stack with:
  - The seen keys.
  - A set of vertices (p, k, q) of the key graph such that (p, k, q) was not traversed.
  - ...

Ideas:

- From a VPA $\mathcal{A}$, construct its key graph.
- Use the key graph to know where to "jump".
  - Potentially, we are in multiple states at the same time (non-determinism).
- During execution, use a stack with:
  - The seen keys.
  - A set of vertices (p, k, q) of the key graph such that (p, k, q) was not traversed.
  - ...

Algorithm is too technical to give here ☺.

Let $d(\mathcal{J})$ denote the depth (number of nested objects and arrays) of the document $\mathcal{J}$.

Theorem 3 (Contribution)

Let $\mathcal{C}$ be a set of constraints over keys Σ_{key} and $\mathcal{A}$ be a VPA that recognizes $\mathcal{C}$. Deciding if a JSON document $\mathcal{J}$ is valid requires

- *space polynomial in $|\mathcal{A}|$, $|\Sigma_{\mathrm{key}}|$, and $d(\mathcal{J})$,*
- *time polynomial in $|\mathcal{J}|$ and $|\mathcal{A}|$, and exponential in $|\Sigma_{\mathrm{key}}|$.*

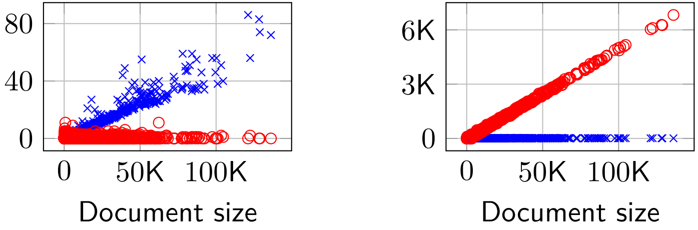
Implemented in Java (thanks to AutomataLib and LearnLib).
We measured the time needed to learn the VPA, and we compared both validation algorithms on six sets of constraints. Four of them come from real-world cases.

	Time	Membership	Equivalence	$\|Q\|$
	9590.3 s	4246085.0	36.4	150.0

(a) Learning.

Time	Computation	Storage	Size
1715 s	11827 kB	419 kB	418

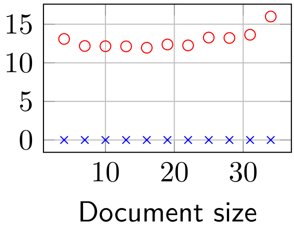
(b) Computation of the key graph.



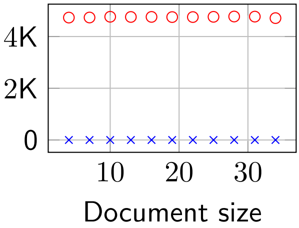
(c) Time usage (ms). (d) Mem. usage (kB).

Figure 3: Results for VIM plugins. $|\Sigma_{\text{key}}| = 16$. Red circles = classical algorithm. Blue crosses = our algorithm.

We use Boolean operations to force the classical algorithm to explore multiple branches, while our algorithm is immediate.



(a) Time usage (ms).



(b) Mem. usage (kB).

Figure 4: Results for a worst case. $|\Sigma_{\text{key}}| = 1$.

References I

Isberner, Malte. "Foundations of active automata learning: an algorithmic perspective". PhD thesis. Technical University Dortmund, Germany, 2015. URL: https://hdl.handle.net/2003/34282.