

Vérifier efficacement un document JSON grâce à un automate

Gaëtan Staquet

Informatique théorique
Département d'informatique
Faculté des Sciences
Université de Mons

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

6 mai 2022

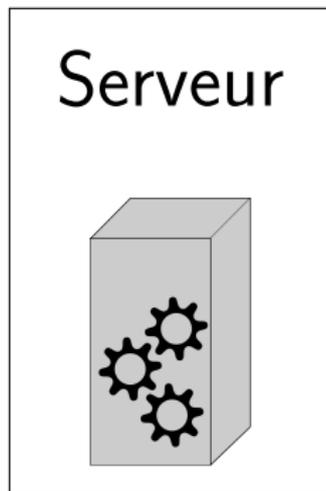
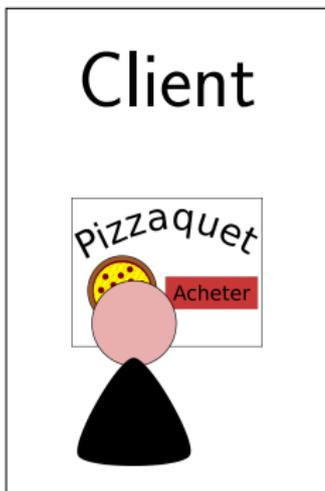


Figure 1 – Un client et un serveur qui communiquent.

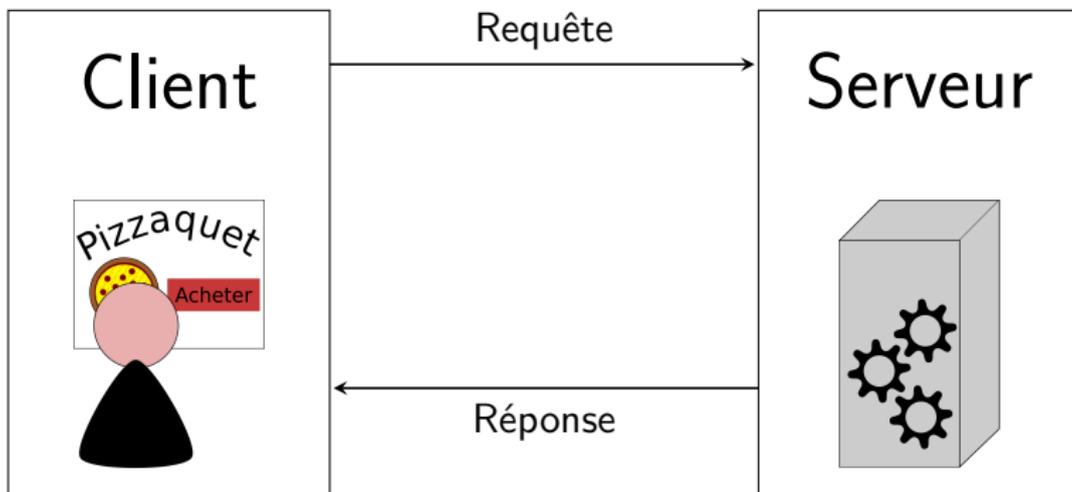


Figure 1 – Un client et un serveur qui communiquent.

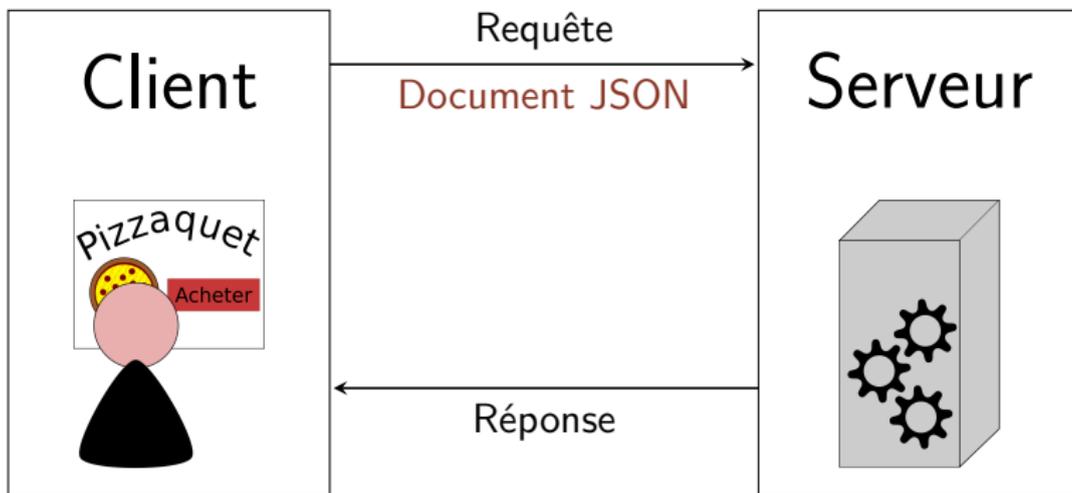


Figure 1 – Un client et un serveur qui communiquent.

```
1 {
2   "orderId": 30,
3   "ingredients": [
4     "olives",
5     "mushrooms",
6     "goat cheese"
7   ],
8   "type": "pizza"
9 }
```

Figure 2 – Exemple d'un document JSON.¹

1. *JSON.org*.

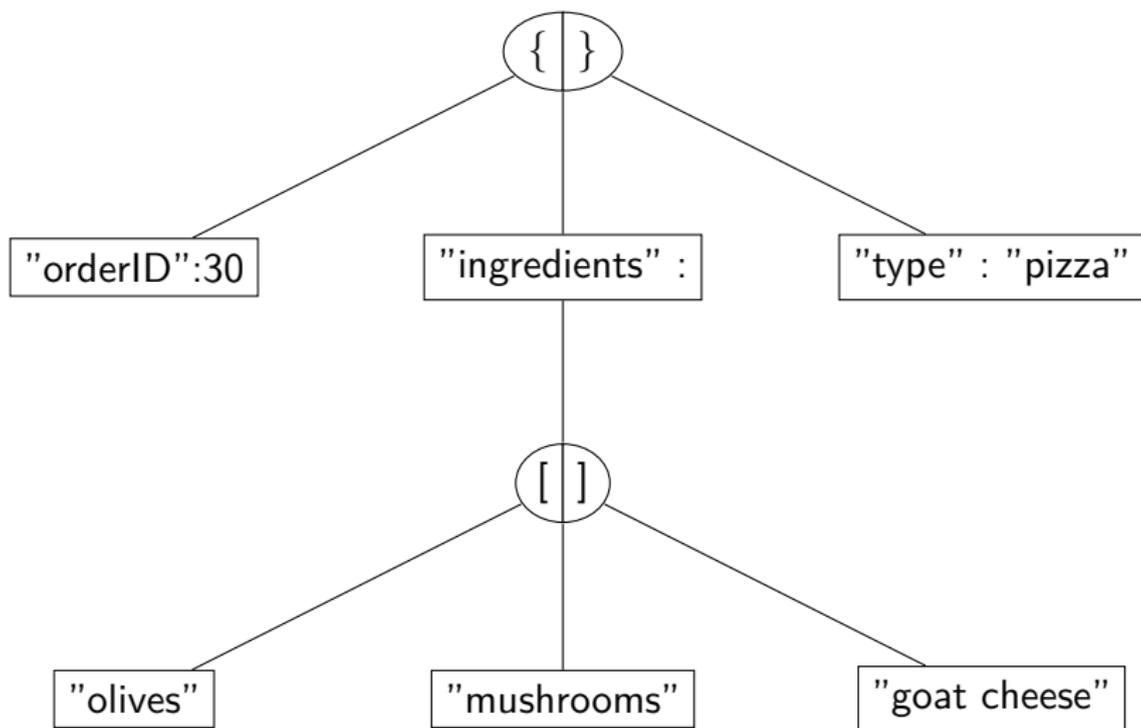


Figure 3 – Un arbre représentant le document JSON.

Type	Symboles	Description
Primitif		Nombres, chaînes de caractère, booléens.
Objet	{ ... }	Collection non-ordonnée de paires clés-valeurs, où chaque clé est une chaîne de caractère et la valeur peut être de n'importe quel type.
Liste	[...]	Collection ordonnée de valeurs. Chaque valeur peut être de n'importe quel type.

Table 1 – Types acceptés dans un document JSON.

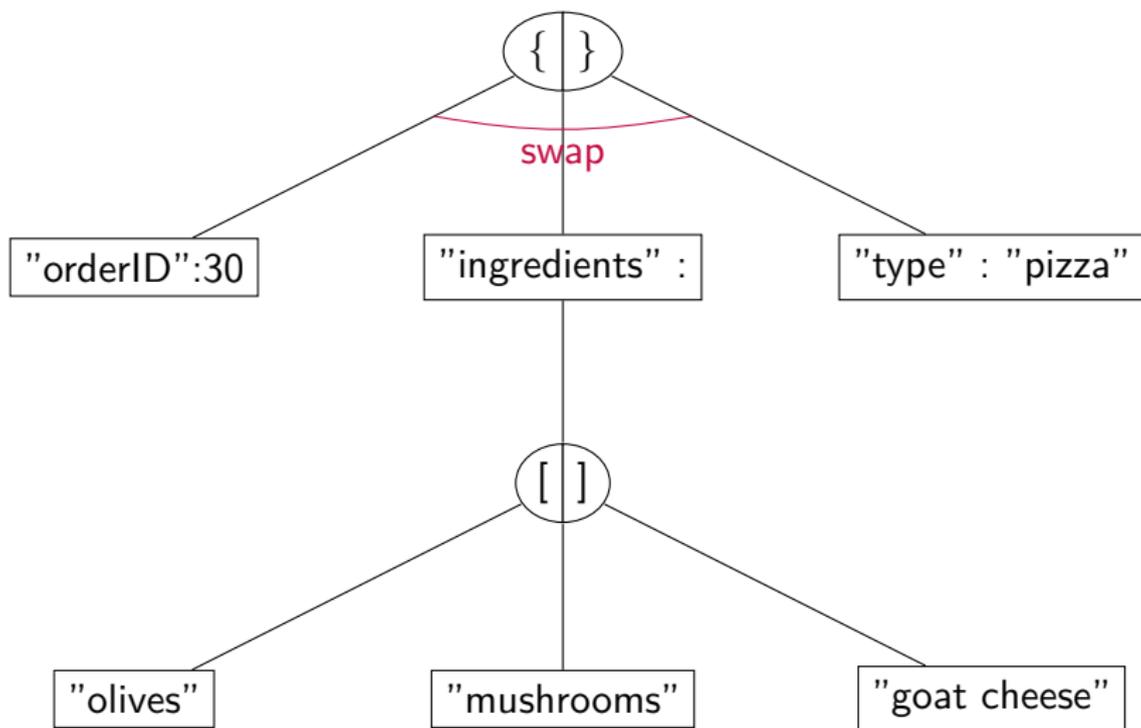


Figure 4 – Arbre représentant un document JSON.

Le serveur doit pouvoir comprendre le document reçu.
↔ On veut que certaines paires clé-valeur soient présentes.

Le serveur doit pouvoir comprendre le document reçu.
↔ On veut que certaines paires clé-valeur soient présentes.

Problème 1

Comment encoder les contraintes de structure qu'un document doit satisfaire ?

Le serveur doit pouvoir comprendre le document reçu.
↔ On veut que certaines paires clé-valeur soient présentes.

Problème 1

Comment encoder les contraintes de structure qu'un document doit satisfaire ?

↔ Via un schéma JSON.²

```
1 {
2   "type": "object",
3   "properties": {"orderId": "integer"},
4   "anyOf": [
5     {
6       "properties": {
7         "type": {"const": "pizza"},
8         "ingredients": "array"
9       }
10    },
11    {
12      "properties": {
13        "type": {"const": "pain à l'ail"},
14        "quantity": "integer"
15      }
16    }
17  ]
18 }
```

Figure 5 – Exemple d'un schéma JSON.

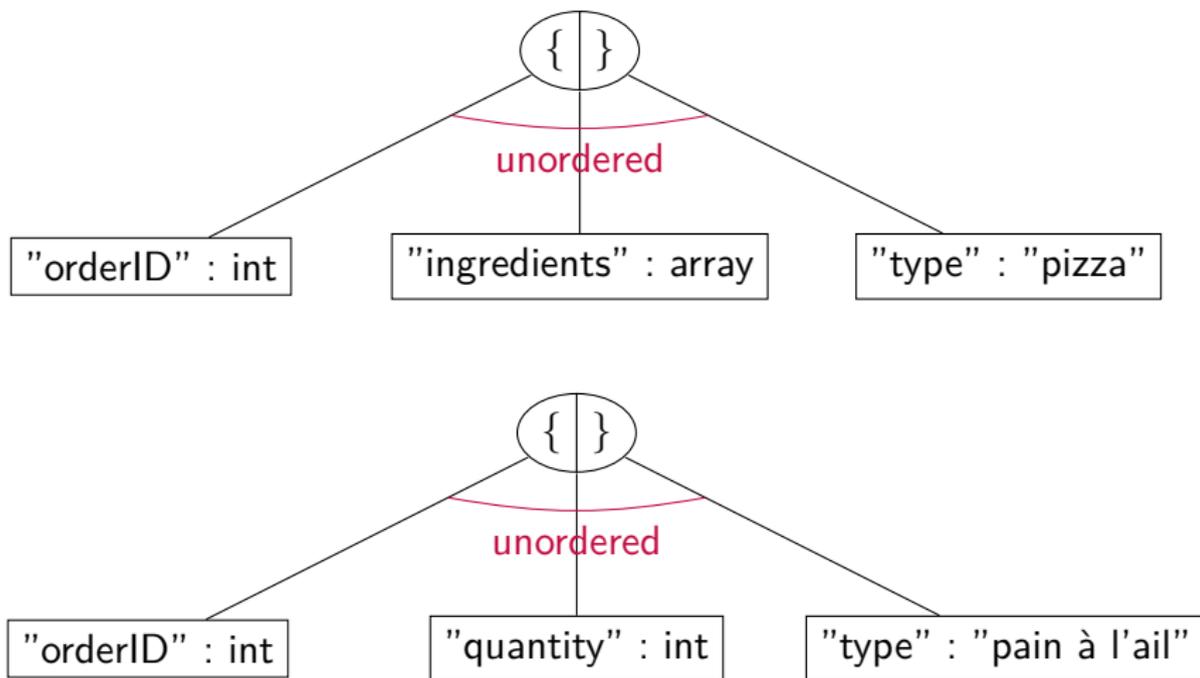


Figure 6 – Les arbres correspondants au schéma.

On veut **valider** si un document donné d satisfait les contraintes données par le schéma \mathcal{S} .

On veut **valider** si un document donné d satisfait les contraintes données par le schéma \mathcal{S} .

Algorithm 2 Algorithme simple pour valider si d satisfait \mathcal{S} .

1 : **tant que** il y a un arbre t non exploré parmi ceux décrits par \mathcal{S}
 faire
2 : **si** d satisfait t **alors**
3 : **retourner** Vrai
4 : **retourner** Faux

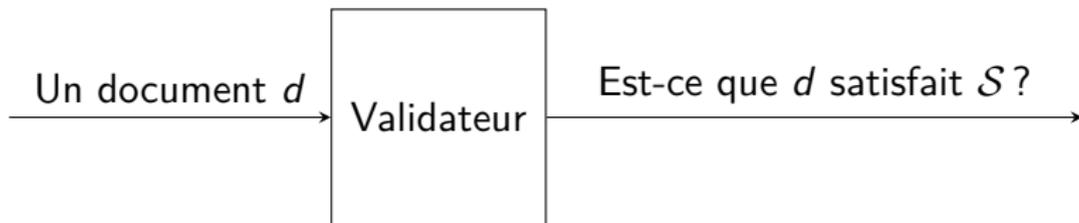


Figure 7 – Vérifier des documents à partir d'un schéma \mathcal{S} .

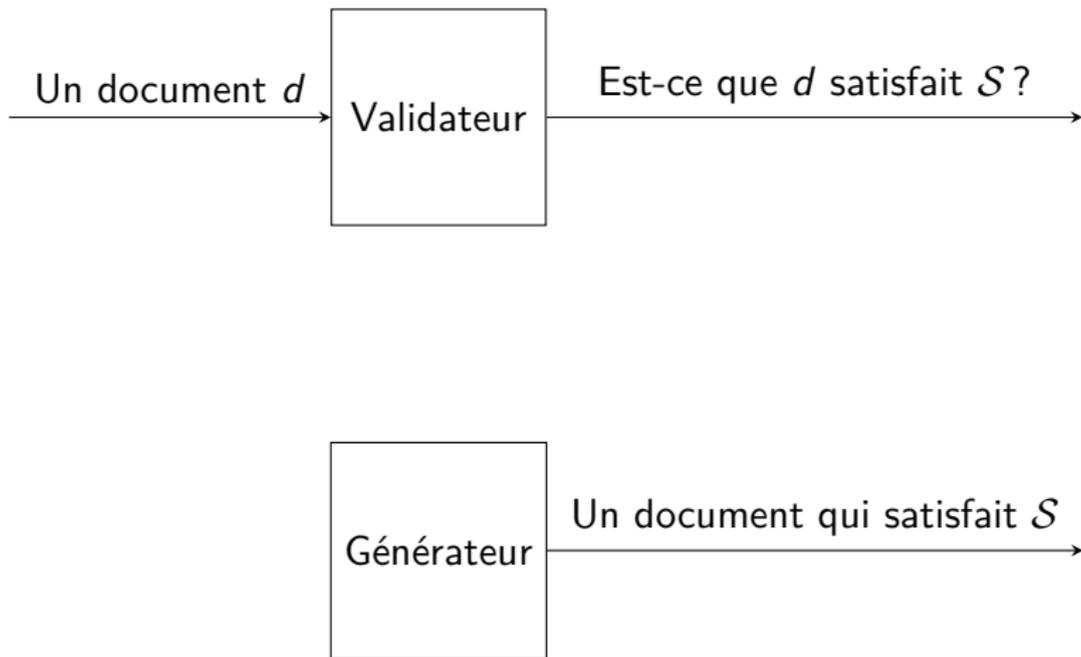


Figure 7 – Vérifier et **générer** des documents à partir d'un schéma \mathcal{S} .

Supposons qu'on reçoive le document petit à petit.

↔ On est dans un cas de **streaming**. On aimerait pouvoir décider si un document est valide sans devoir attendre d'avoir tout reçu.

Supposons qu'on reçoive le document petit à petit.

↔ On est dans un cas de **streaming**. On aimerait pouvoir décider si un document est valide sans devoir attendre d'avoir tout reçu.

On veut donc éviter de devoir choisir un arbre pour, plus tard, devoir faire un autre choix.

Supposons qu'on reçoive le document petit à petit.

↪ On est dans un cas de **streaming**. On aimerait pouvoir décider si un document est valide sans devoir attendre d'avoir tout reçu.

On veut donc éviter de devoir choisir un arbre pour, plus tard, devoir faire un autre choix.

Problème 2

Dans un contexte de streaming, comment vérifier le document sans devoir attendre d'avoir tout reçu ?

Supposons qu'on reçoive le document petit à petit.

↔ On est dans un cas de **streaming**. On aimerait pouvoir décider si un document est valide sans devoir attendre d'avoir tout reçu.

On veut donc éviter de devoir choisir un arbre pour, plus tard, devoir faire un autre choix.

Problème 2

Dans un contexte de streaming, comment vérifier le document sans devoir attendre d'avoir tout reçu ?

↔ Avec la théorie des **automates** !

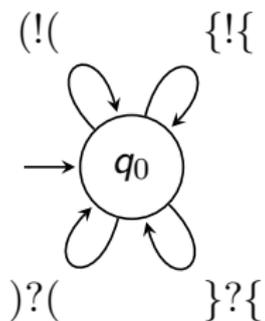
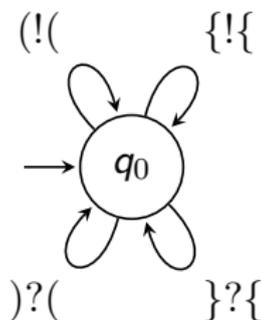


Figure 8 – Un automate à pile.³

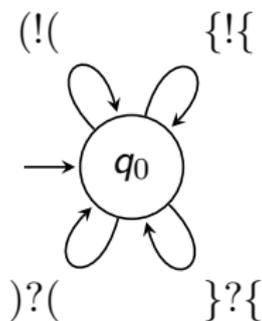
3. HOPCROFT et ULLMAN, *Introduction to Automata Theory, Languages and Computation*, 2000.



Lisons le mot $\{ \{ (()) \} \}$.

Figure 8 – Un automate à pile.³

3. HOPCROFT et ULLMAN, *Introduction to Automata Theory, Languages and Computation*, 2000.

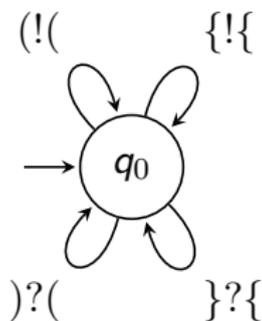


Lisons le mot $\{((())\}$.



Figure 8 – Un **automate à pile**.³

3. HOPCROFT et ULLMAN, *Introduction to Automata Theory, Languages and Computation*, 2000.

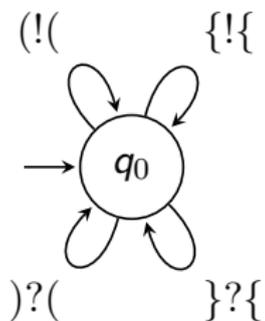


Lisons le mot $\{((())\}$.



Figure 8 – Un automate à pile.³

3. HOPCROFT et ULLMAN, *Introduction to Automata Theory, Languages and Computation*, 2000.



Lisons le mot $\{ \{ (()) \} \}$.

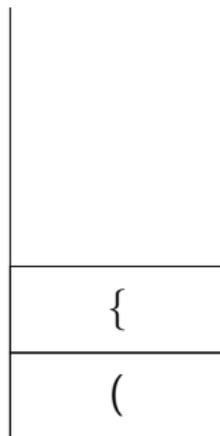
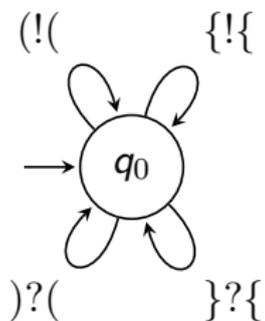


Figure 8 – Un **automate à pile**.³

3. HOPCROFT et ULLMAN, *Introduction to Automata Theory, Languages and Computation*, 2000.



Lisons le mot $\{ \{ (()) \} \}$.

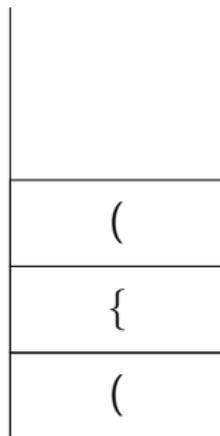
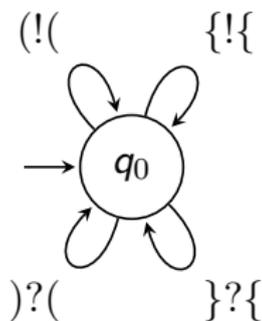


Figure 8 – Un automate à pile.³

3. HOPCROFT et ULLMAN, *Introduction to Automata Theory, Languages and Computation*, 2000.



Lisons le mot $\{((())\}$.

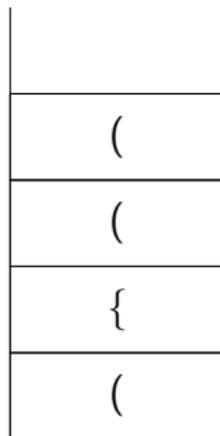
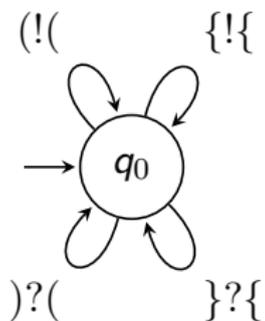


Figure 8 – Un **automate à pile**.³

3. HOPCROFT et ULLMAN, *Introduction to Automata Theory, Languages and Computation*, 2000.



Lisons le mot $\{((())\})$.

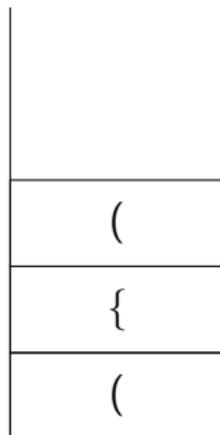
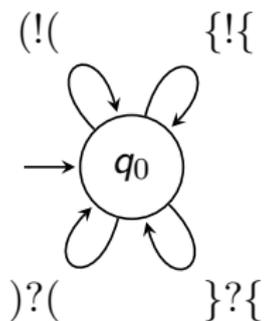


Figure 8 – Un **automate à pile**.³

3. HOPCROFT et ULLMAN, *Introduction to Automata Theory, Languages and Computation*, 2000.



Lisons le mot $\{ \{ (()) \} \}$.

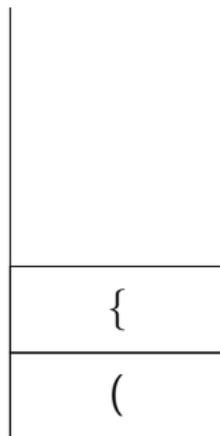
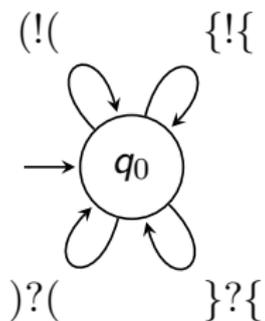


Figure 8 – Un automate à pile.³

3. HOPCROFT et ULLMAN, *Introduction to Automata Theory, Languages and Computation*, 2000.

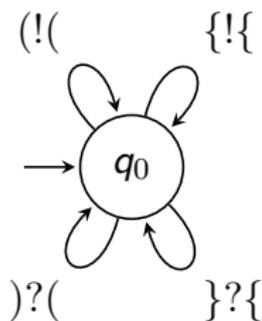


Lisons le mot $\{((())\}$.



Figure 8 – Un automate à pile.³

3. HOPCROFT et ULLMAN, *Introduction to Automata Theory, Languages and Computation*, 2000.



Lisons le mot $\{((())\}$.

Le mot est correct.



Figure 8 – Un automate à pile.³

3. HOPCROFT et ULLMAN, *Introduction to Automata Theory, Languages and Computation*, 2000.

On peut utiliser un automate à pile pour vérifier si un document est correct pour un schéma S .

Problème 3

Comment construire cet automate à pile ?

On peut utiliser un automate à pile pour vérifier si un document est correct pour un schéma \mathcal{S} .

Problème 3

Comment construire cet automate à pile ?

↔ Avec l'apprentissage automatique d'automates !

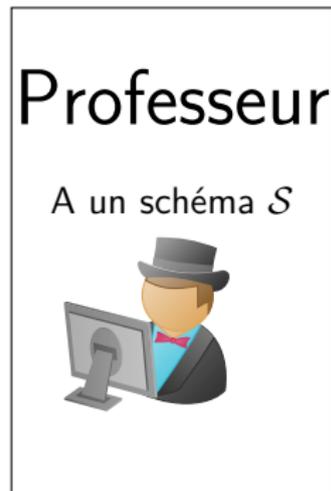


Figure 9 – Adaptation du framework d'Angluin pour les documents JSON.⁴

4. ANGLUIN, « Learning Regular Sets from Queries and Counterexamples », 1987.

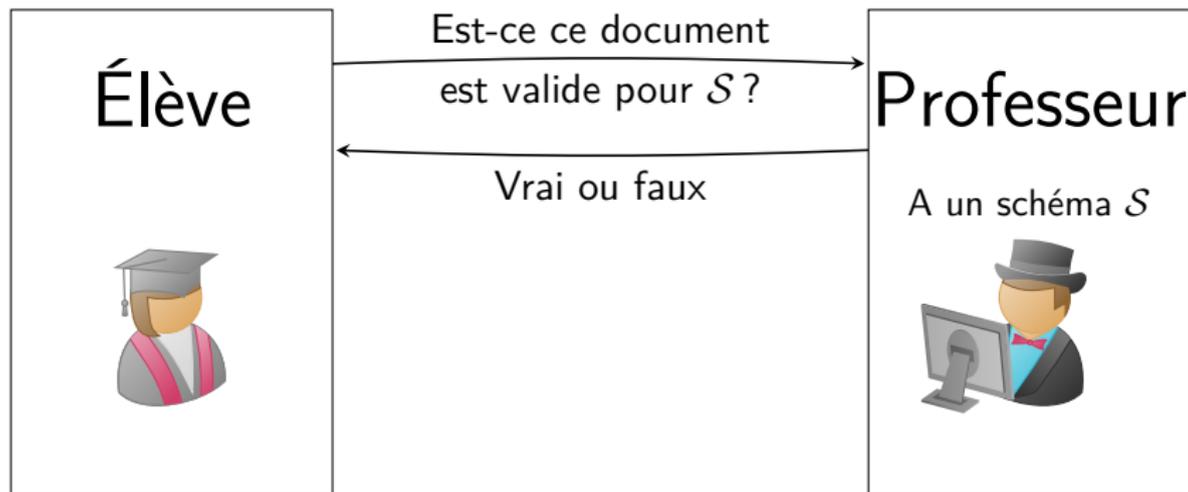


Figure 9 – Adaptation du framework d'Angluin pour les documents JSON.⁴

4. ANGLUIN, « Learning Regular Sets from Queries and Counterexamples », 1987.

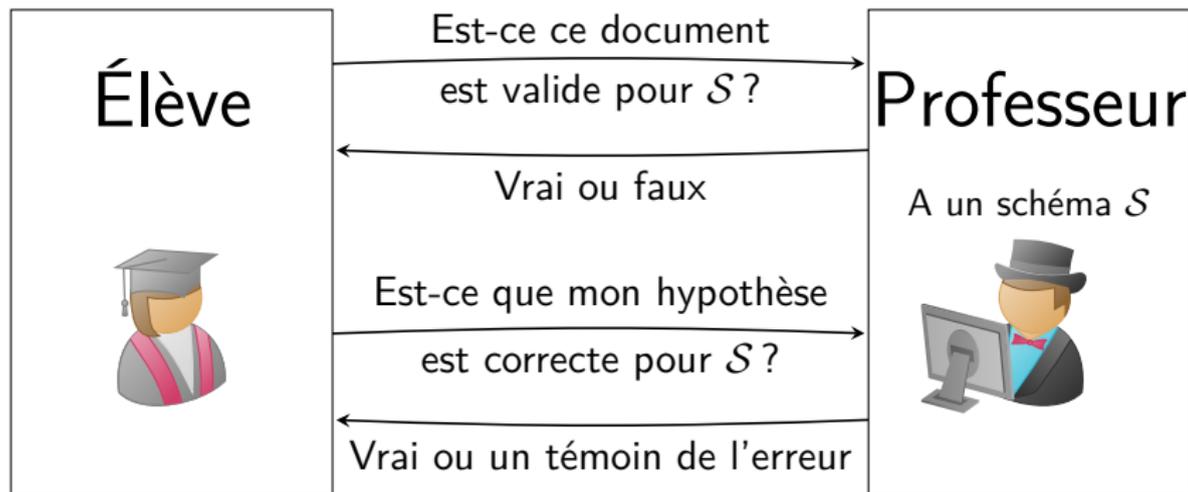


Figure 9 – Adaptation du framework d'Angluin pour les documents JSON.⁴

4. ANGLUIN, « Learning Regular Sets from Queries and Counterexamples », 1987.

On peut apprendre un automate à pile à partir d'un schéma.⁵
Est-ce qu'on a la solution parfaite ?

5. ISBERNER, « Foundations of active automata learning : an algorithmic perspective », 2015.

On peut apprendre un automate à pile à partir d'un schéma.⁵
Est-ce qu'on a la solution parfaite ?

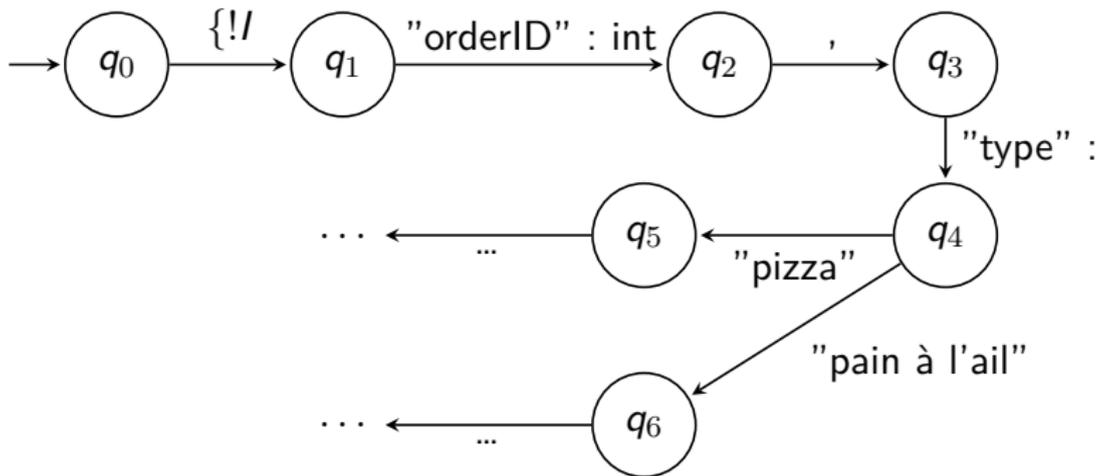


Figure 10 – Un automate à pile pour un document JSON.

5. ISBERNER, « Foundations of active automata learning : an algorithmic perspective », 2015.

On peut apprendre un automate à pile à partir d'un schéma.⁵
Est-ce qu'on a la solution parfaite ?

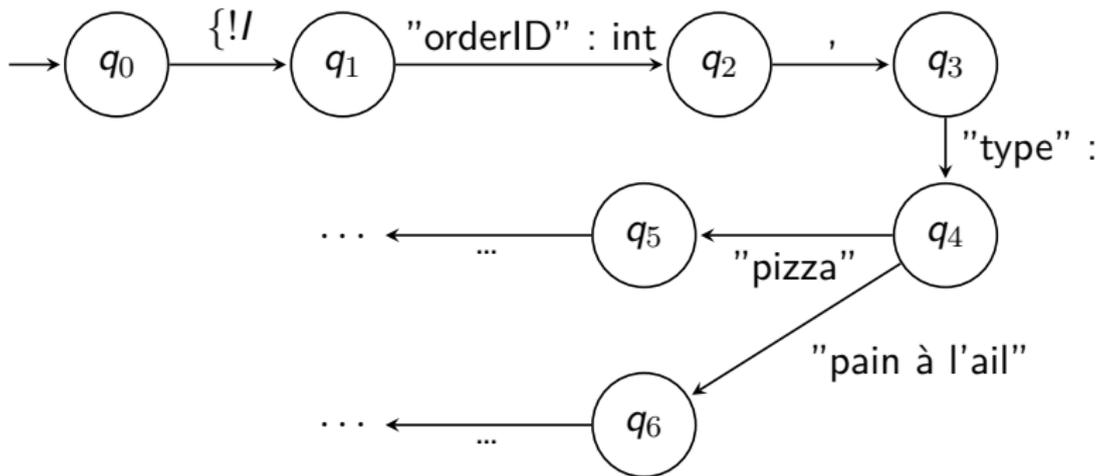


Figure 10 – Un automate à pile pour un document JSON.

Les automates à pile attendent un ordre fixé pour les paires clé-valeur... 😞

5. ISBERNER, « Foundations of active automata learning : an algorithmic perspective », 2015.

Problème 4

Comment faire en sorte qu'un automate à pile accepte un document quelque soit l'ordre des paires clé-valeur ?

↔ Travail en cours.

En conclusion :

- ▶ Les concepts étudiés en informatique théorique ont des applications en pratique.
- ▶ Une approche théorique à un problème pratique peut apporter de nouvelles solutions.
- ▶ Mais cela demande du temps.

Références I

-  ANGLUIN, Dana. « Learning Regular Sets from Queries and Counterexamples ». In : *Inf. Comput.* 75.2 (1987), p. 87-106. DOI : [10.1016/0890-5401\(87\)90052-6](https://doi.org/10.1016/0890-5401(87)90052-6). URL : [https://doi.org/10.1016/0890-5401\(87\)90052-6](https://doi.org/10.1016/0890-5401(87)90052-6).
-  HOPCROFT, John E. et Jeffrey D. ULLMAN. *Introduction to Automata Theory, Languages and Computation, Second Edition*. Addison-Wesley, 2000.
-  ISBERNER, Malte. « Foundations of active automata learning : an algorithmic perspective ». Thèse de doct. Technical University Dortmund, Germany, 2015. URL : <https://hdl.handle.net/2003/34282>.
-  *JSON Schema*. Official website. URL : <https://json-schema.org>.
-  *JSON.org*. Website describing the structure of a JSON document. URL : <https://www.json.org>.