

Vérifier un système informatique grâce à un automate

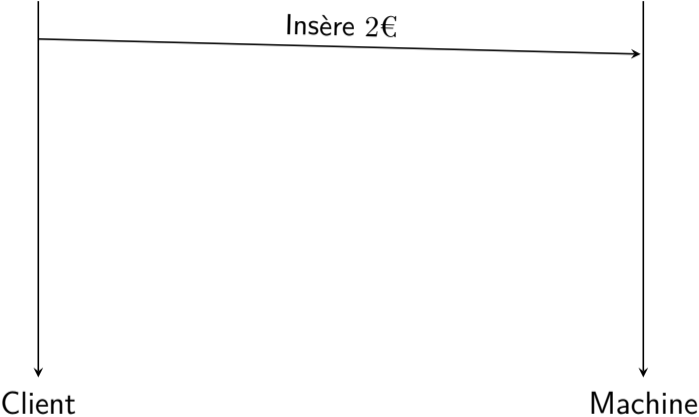
Gaëtan Staquet

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

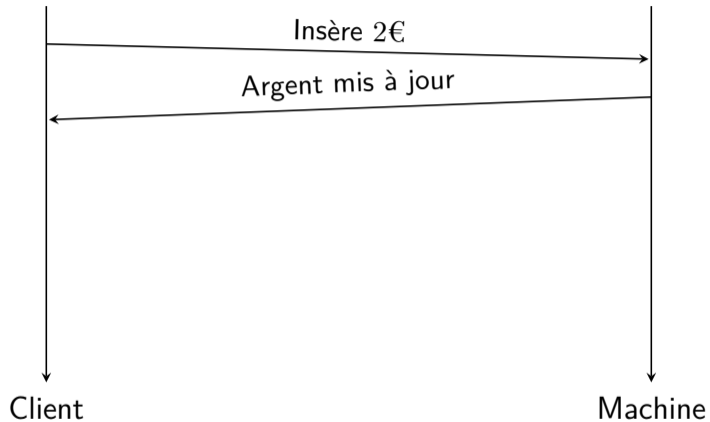
Formal Techniques in Software Engineering
Computer Science Department
Science Faculty
University of Antwerp

24 octobre 2022

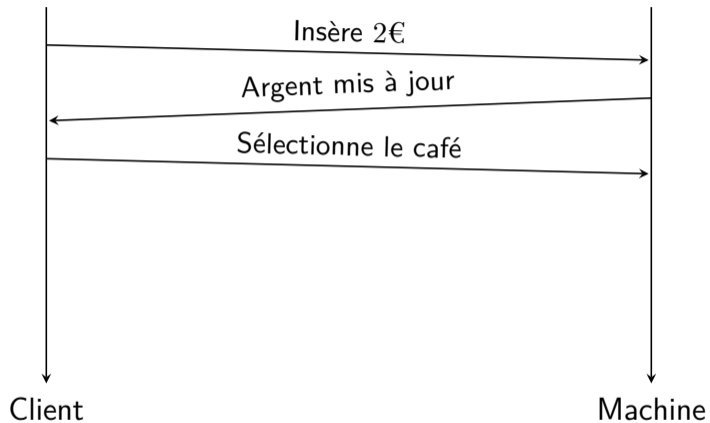
Machine à café – Bonne exécution



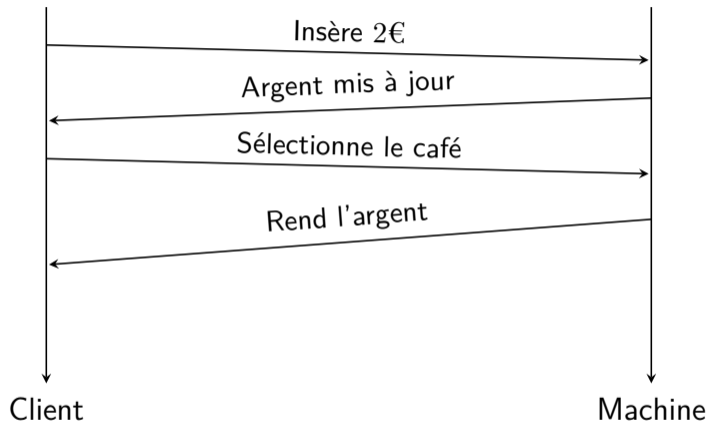
Machine à café – Bonne exécution



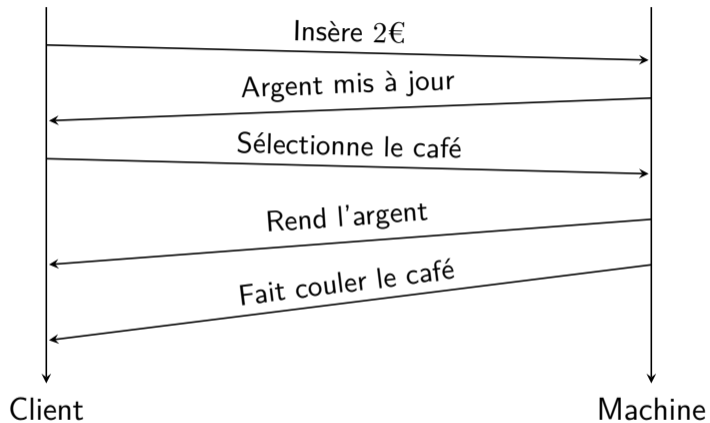
Machine à café – Bonne exécution



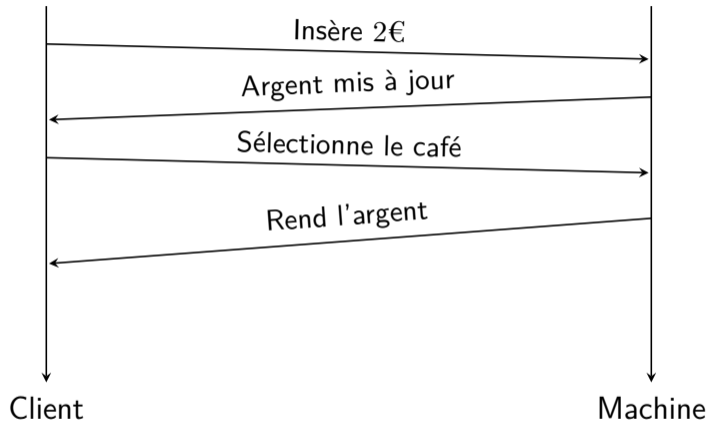
Machine à café – Bonne exécution



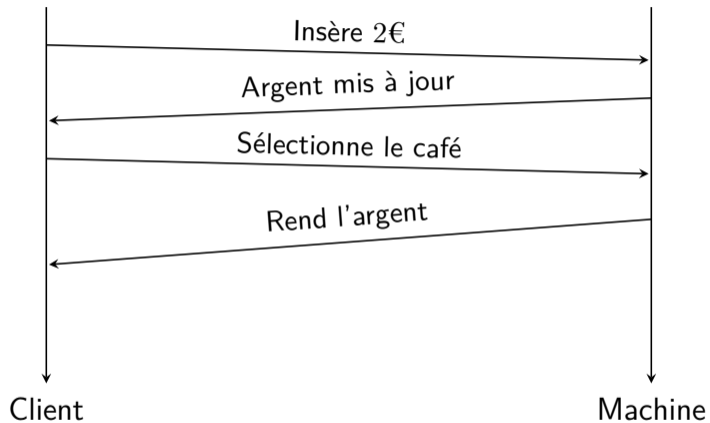
Machine à café – Bonne exécution



Machine à café – Erreur



Machine à café – Erreur



Comment détecter l'erreur le plus tôt possible ?

Tests unitaires ?

Tests unitaires ?

- ▶ Demande d'implémenter les tests « à la main ».

Tests unitaires ?

- ▶ Demande d'implémenter les tests « à la main ».
- ▶ Risque d'oublier des cas importants.

Tests unitaires ?

- ▶ Demande d'implémenter les tests « à la main ».
- ▶ Risque d'oublier des cas importants.
- ▶ On ne peut pas tout tester.

Tests unitaires ?

- ▶ Demande d'implémenter les tests « à la main ».
- ▶ Risque d'oublier des cas importants.
- ▶ On ne peut pas tout tester.

↪ Ne prouve pas que le système est correct.

Tests unitaires ?

- ▶ Demande d'implémenter les tests « à la main ».
- ▶ Risque d'oublier des cas importants.
- ▶ On ne peut pas tout tester.

↪ Ne prouve pas que le système est correct.

On va exploiter les **méthodes formelles**.

Détecter des erreurs

Tests unitaires ?

- ▶ Demande d'implémenter les tests « à la main ».
- ▶ Risque d'oublier des cas importants.
- ▶ On ne peut pas tout tester.

↪ Ne prouve pas que le système est correct.

On va exploiter les **méthodes formelles**.

Idée :

- ▶ **Construire** un modèle \mathcal{M} du système.
- ▶ **Vérifier** si \mathcal{M} satisfait les propriétés qu'on désire.

Détecter des erreurs

Tests unitaires ?

- ▶ Demande d'implémenter les tests « à la main ».
- ▶ Risque d'oublier des cas importants.
- ▶ On ne peut pas tout tester.

↪ Ne prouve pas que le système est correct.

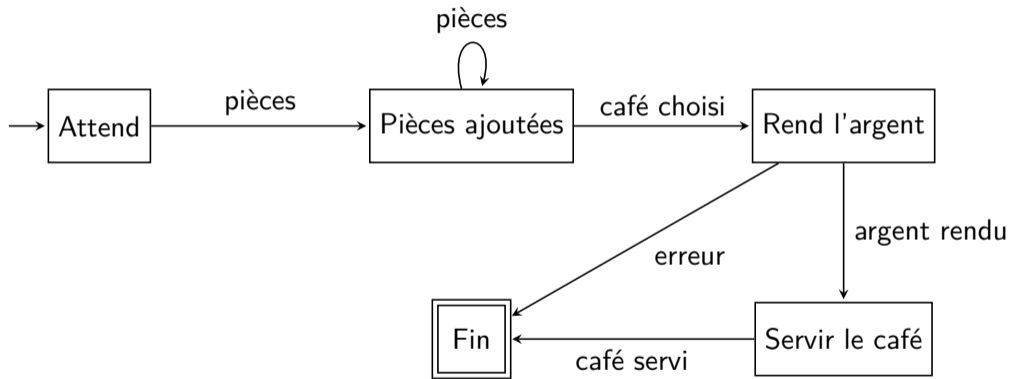
On va exploiter les **méthodes formelles**.

Idée :

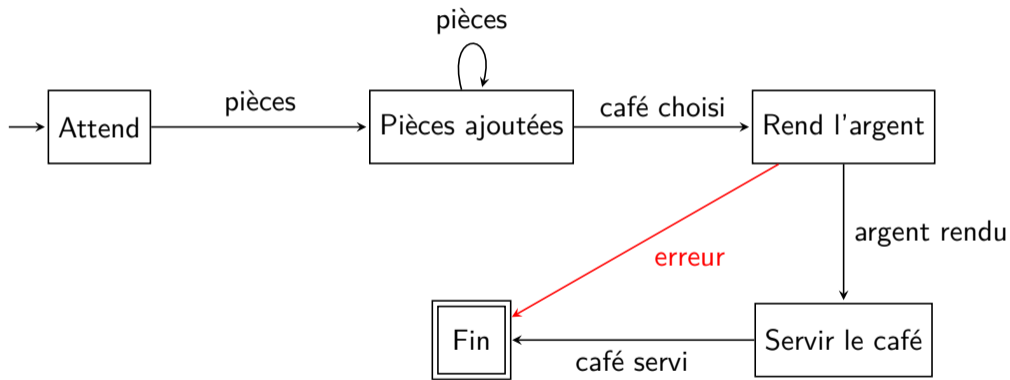
- ▶ **Construire** un modèle \mathcal{M} du système.
- ▶ **Vérifier** si \mathcal{M} satisfait les propriétés qu'on désire.

Ici, on s'intéresse à la **construction** du modèle.

Un modèle pour la machine à café



Un modèle pour la machine à café



Quel modèle ?

Un **alphabet**, noté Σ , est un ensemble fini et non-vide de **symboles**.

Exemple 1

$\Sigma = \{a, b\}$ est un alphabet.

Quel modèle ?

Un **alphabet**, noté Σ , est un ensemble fini et non-vide de **symboles**.

Un **mot** $w = a_1 a_2 \dots a_n$ ($n \in \mathbb{N}$) sur un alphabet Σ est une séquence finie de symboles, $a_i \in \Sigma$. Le **mot vide** est dénoté ϵ .

Exemple 1

$\Sigma = \{a, b\}$ est un alphabet.

$w = ababb$ est un mot sur Σ .

Quel modèle ?

Un **alphabet**, noté Σ , est un ensemble fini et non-vide de **symboles**.

Un **mot** $w = a_1 a_2 \dots a_n$ ($n \in \mathbb{N}$) sur un alphabet Σ est une séquence finie de symboles, $a_i \in \Sigma$. Le **mot vide** est dénoté ε .

Un **langage** L sur un alphabet Σ est un **ensemble de mots**.

Exemple 1

$\Sigma = \{a, b\}$ est un alphabet.

$w = ababb$ est un mot sur Σ .

$L' = \{\varepsilon, a, b\}$ et $L = \{w \mid w \text{ a un nombre pair de } a \text{ et un nombre impair de } b\}$ sont deux langages sur Σ .

Quel modèle ?

Un **automate fini déterministe** (DFA) est un tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ avec

- ▶ Σ un alphabet ;

Quel modèle ?

Un **automate fini déterministe** (DFA) est un tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ avec

- ▶ Σ un alphabet ;
- ▶ Q un ensemble fini d'**états** ;

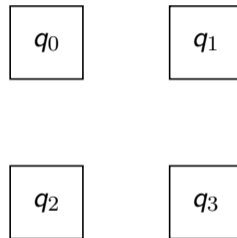


Figure 1 – Un DFA \mathcal{A} .

Quel modèle ?

Un **automate fini déterministe** (DFA) est un tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ avec

- ▶ Σ un alphabet ;
- ▶ Q un ensemble fini d'**états** ;
- ▶ $\delta : (Q \times \Sigma) \rightarrow Q$ une **fonction de transition** ;

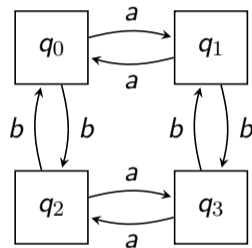


Figure 1 – Un DFA \mathcal{A} .

Quel modèle ?

Un **automate fini déterministe** (DFA) est un tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ avec

- ▶ Σ un alphabet ;
- ▶ Q un ensemble fini d'**états** ;
- ▶ $\delta : (Q \times \Sigma) \rightarrow Q$ une **fonction de transition** ;
- ▶ $q_0 \in Q$ l'**état initial** ;

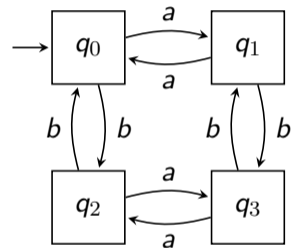


Figure 1 – Un DFA \mathcal{A} .

Quel modèle ?

Un **automate fini déterministe** (DFA) est un tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ avec

- ▶ Σ un alphabet ;
- ▶ Q un ensemble fini d'**états** ;
- ▶ $\delta : (Q \times \Sigma) \rightarrow Q$ une **fonction de transition** ;
- ▶ $q_0 \in Q$ l'**état initial** ;
- ▶ $F \subseteq Q$ l'ensemble des **états finaux**.

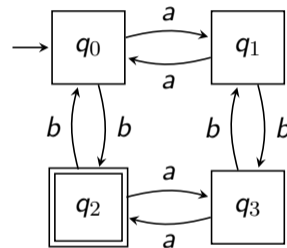


Figure 1 – Un DFA \mathcal{A} .

Quel modèle ?

Soit $w = a_1 a_2 \dots, a_n \in \Sigma^*$. L'exécution de \mathcal{A} sur w est la séquence d'états

$$p_1 \xrightarrow{a_1} p_2 \xrightarrow{a_2} p_3 \xrightarrow{a_3} \dots \xrightarrow{a_n} p_{n+1}$$

telle que $p_1 = q_0$ et $\forall i, \delta(p_i, a_i) = p_{i+1}$.

Exemple 2

Soit $w = ababb$. L'exécution correspondante est

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_3 \xrightarrow{a} q_2 \xrightarrow{b} q_0 \xrightarrow{b} q_2.$$

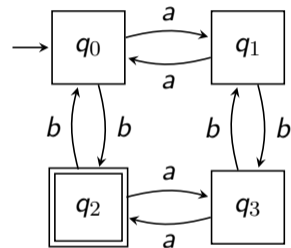


Figure 1 – Un DFA \mathcal{A} .

Quel modèle ?

Soit $w = a_1 a_2 \dots, a_n \in \Sigma^*$. L'exécution de \mathcal{A} sur w est la séquence d'états

$$p_1 \xrightarrow{a_1} p_2 \xrightarrow{a_2} p_3 \xrightarrow{a_3} \dots \xrightarrow{a_n} p_{n+1}$$

telle que $p_1 = q_0$ et $\forall i, \delta(p_i, a_i) = p_{i+1}$.

Si $p_{n+1} \in F$, alors w est accepté par \mathcal{A} .

Exemple 2

Soit $w = ababb$. L'exécution correspondante est

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_3 \xrightarrow{a} q_2 \xrightarrow{b} q_0 \xrightarrow{b} q_2,$$

et w est accepté par \mathcal{A} .

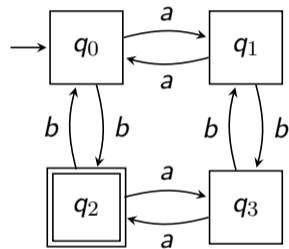


Figure 1 – Un DFA \mathcal{A} .

Quel modèle ?

Le langage de \mathcal{A} est l'ensemble des mots acceptés, i.e.,

$$\mathcal{L}(\mathcal{A}) = \{w \mid \exists p \in F, q_0 \xrightarrow{w} p\}.$$

Exemple 3

Le langage de \mathcal{A} est

$$\mathcal{L}(\mathcal{A}) = \{w \mid w \text{ a un nombre pair de } a \text{ et un nombre impair de } b\}.$$

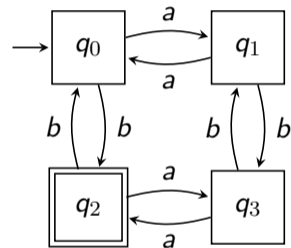


Figure 1 – Un DFA \mathcal{A} .

Table infinie

Soit $L = \{w \mid w \text{ a un nombre pair de } a \text{ et un nombre impair de } b\}$.

Soit $u \in \Sigma^*$. Pour tout $w \in \Sigma^*$, on vérifie si $uw \in L$.

On construit une table dont les lignes sont indexées par les u et les colonnes par les w .

Table infinie

Soit $L = \{w \mid w \text{ a un nombre pair de } a \text{ et un nombre impair de } b\}$.

	ε	a	b	aa	ab	ba	bb	\dots
ε	0	0	1	0	0	0	0	\dots
a	0	0	0	0	1	1	0	\dots
b	1	0	0	1	0	0	1	\dots
aa	0	0	1	0	0	0	0	\dots
ab	0	1	0	0	0	0	0	\dots
ba	0	1	0	0	0	0	0	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

Table infinie

Soit $L = \{w \mid w \text{ a un nombre pair de } a \text{ et un nombre impair de } b\}$.

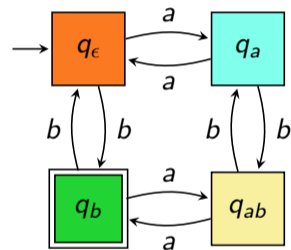
	ε	a	b	aa	ab	ba	bb	\dots
ε	0	0	1	0	0	0	0	\dots
a	0	0	0	0	1	1	0	\dots
b	1	0	0	1	0	0	1	\dots
aa	0	0	1	0	0	0	0	\dots
ab	0	1	0	0	0	0	0	\dots
ba	0	1	0	0	0	0	0	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

La table contient en vérité quatre lignes différentes.

Table infinie

Soit $L = \{w \mid w \text{ a un nombre pair de } a \text{ et un nombre impair de } b\}$.

	ϵ	a	b	aa	ab	ba	bb	...
ϵ	0	0	1	0	0	0	0	...
a	0	0	0	0	1	1	0	...
b	1	0	0	1	0	0	1	...
aa	0	0	1	0	0	0	0	...
ab	0	1	0	0	0	0	0	...
ba	0	1	0	0	0	0	0	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮



La table contient en vérité quatre lignes différentes.

Table infinie

Soit $L = \{w \mid w \text{ a un nombre pair de } a \text{ et un nombre impair de } b\}$.

	ε	a	b	aa	ab	ba	bb	\dots
ε	0	0	1	0	0	0	0	\dots
a	0	0	0	0	1	1	0	\dots
b	1	0	0	1	0	0	1	\dots
aa	0	0	1	0	0	0	0	\dots
ab	0	1	0	0	0	0	0	\dots
ba	0	1	0	0	0	0	0	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

La table contient en vérité quatre lignes différentes.

\hookrightarrow On peut se contenter d'une sous-table finie.

Comment apprendre une table ?

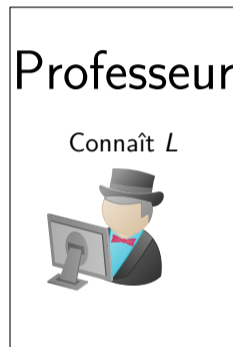


Figure 2 – Le framework d'Angluin.¹

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

Comment apprendre une table ?

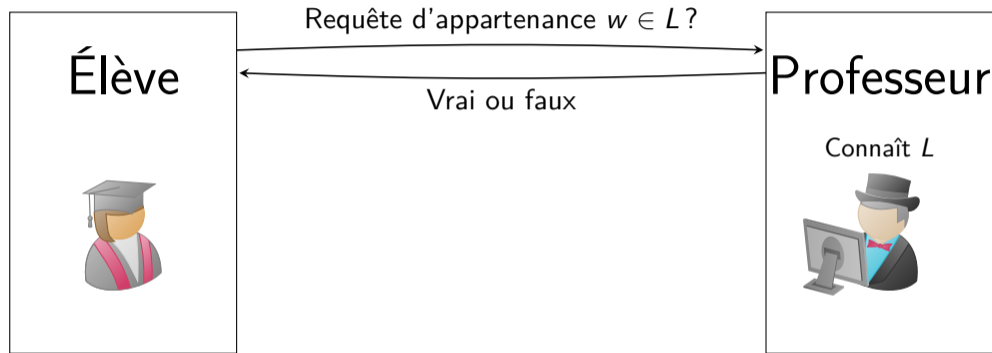


Figure 2 – Le framework d'Angluin.¹

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

Comment apprendre une table ?

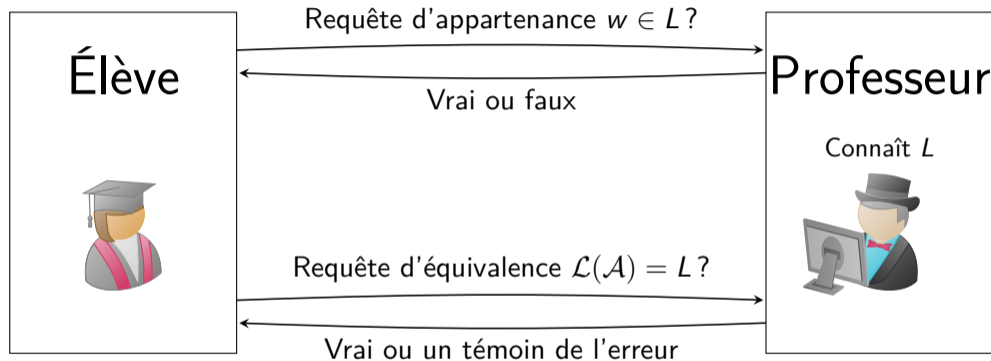


Figure 2 – Le framework d'Angluin. ¹

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

Comment fonctionne le professeur en pratique ?

- ▶ **Requêtes d'appartenance** : exécuter le programme sur w et donner la réponse.

Comment fonctionne le professeur en pratique ?

- ▶ **Requêtes d'appartenance** : exécuter le programme sur w et donner la réponse.
- ▶ **Requêtes d'équivalence** :
 - ▶ Si on peut manipuler le programme comme une **boîte noire**, alors on peut approximer les requêtes d'équivalence.

Comment fonctionne le professeur en pratique ?

- ▶ **Requêtes d'appartenance** : exécuter le programme sur w et donner la réponse.
- ▶ **Requêtes d'équivalence** :
 - ▶ Si on peut manipuler le programme comme une **boîte noire**, alors on peut approximer les requêtes d'équivalence.
 - ▶ Si on a accès à l'intérieur du programme (**boîte blanche**), alors les requêtes d'équivalence peuvent être plus précises.

Comment fonctionne le professeur en pratique ?

- ▶ **Requêtes d'appartenance** : exécuter le programme sur w et donner la réponse.
- ▶ **Requêtes d'équivalence** :
 - ▶ Si on peut manipuler le programme comme une **boîte noire**, alors on peut approximer les requêtes d'équivalence.
 - ▶ Si on a accès à l'intérieur du programme (**boîte blanche**), alors les requêtes d'équivalence peuvent être plus précises.
 - ▶ On peut mixer les deux (**boîte grise**).

Comment fonctionne le professeur en pratique ?

- ▶ **Requêtes d'appartenance** : exécuter le programme sur w et donner la réponse.
- ▶ **Requêtes d'équivalence** :
 - ▶ Si on peut manipuler le programme comme une **boîte noire**, alors on peut approximer les requêtes d'équivalence.
 - ▶ Si on a accès à l'intérieur du programme (**boîte blanche**), alors les requêtes d'équivalence peuvent être plus précises.
 - ▶ On peut mixer les deux (**boîte grise**).

↔ Cela dépend du problème étudié.

Documents JSON

```
{  
  "titre": "Vérification par automates",  
  "lieu": {  
    "ville": "Mons",  
    "pays": "Belgique"  
  },  
  "inscrits": [9, 0, 13, 5, 1, 14]  
}
```

Documents JSON

```
{  
  "titre": "Vérification par automates",  
  "lieu": {  
    "ville": "Mons",  
    "pays": "Belgique"  
  },  
  "inscrits": [9, 0, 13, 5, 1, 14]  
}
```

- Un **objet** est une collection **non-ordonnée** de paires clé-valeur.

Documents JSON

```
{  
  "titre": "Vérification par automates",  
  "lieu": {  
    "ville": "Mons",  
    "pays": "Belgique"  
  },  
  "inscrits": [9, 0, 13, 5, 1, 14]  
}
```

- ▶ Un objet est une collection **non-ordonnée** de paires clé-valeur.
- ▶ Un **tableau** est une collection **ordonnée** de valeurs.

Documents JSON

```
{  
  "titre": "Vérification par automates",  
  "lieu": {  
    "ville": "Mons",  
    "pays": "Belgique"  
  },  
  "inscrits": [9, 0, 13, 5, 1, 14]  
}
```

- ▶ Un objet est une collection **non-ordonnée** de paires clé-valeur.
- ▶ Un tableau est une collection **ordonnée** de valeurs.

On veut vérifier que le document satisfait certaines contraintes.

Documents JSON

```
{  
  "titre": "Vérification par automates",  
  "lieu": {  
    "ville": "Mons",  
    "pays": "Belgique"  
  },  
  "inscrits": [9, 0, 13, 5, 1, 14]  
}
```

"titre" \mapsto chaîne de caractère

"lieu" \mapsto objet tel que

"ville" \mapsto chaîne de caractère

"pays" \mapsto chaîne de caractère

"inscrits" \mapsto tableau de 6 entiers

- ▶ Un objet est une collection **non-ordonnée** de paires clé-valeur.
- ▶ Un tableau est une collection **ordonnée** de valeurs.

On veut vérifier que le document satisfait certaines contraintes.

Documents JSON

```
{  
  "titre": "Vérification par automates",  
  "lieu": {  
    "ville": "Mons",  
    "pays": "Belgique"  
  },  
  "inscrits": [9, 0, 13, 5, 1, 14]  
}
```

"titre" \mapsto chaîne de caractère

"lieu" \mapsto objet tel que

"ville" \mapsto chaîne de caractère

"pays" \mapsto chaîne de caractère

"inscrits" \mapsto tableau de 6 entiers

- ▶ Un objet est une collection **non-ordonnée** de paires clé-valeur.
- ▶ Un tableau est une collection **ordonnée** de valeurs.

On veut vérifier que le document satisfait certaines contraintes. Notre approche² :

2. BRUYÈRE, PÉREZ et STAQUET, *Validating JSON Documents with Learned VPAs*, 2022.

Documents JSON

```
{  
  "titre": "Vérification par automates",  
  "lieu": {  
    "ville": "Mons",  
    "pays": "Belgique"  
  },  
  "inscrits": [9, 0, 13, 5, 1, 14]  
}
```

"titre" \mapsto chaîne de caractère

"lieu" \mapsto objet tel que

"ville" \mapsto chaîne de caractère

"pays" \mapsto chaîne de caractère

"inscrits" \mapsto tableau de 6 entiers

- ▶ Un objet est une collection **non-ordonnée** de paires clé-valeur.
- ▶ Un tableau est une collection **ordonnée** de valeurs.

On veut vérifier que le document satisfait certaines contraintes. Notre approche² :

- ▶ On apprend un automate \mathcal{A} avec un ordre sur les clés.

2. BRUYÈRE, PÉREZ et STAQUET, *Validating JSON Documents with Learned VPAs*, 2022.

Documents JSON

```
{  
  "titre": "Vérification par automates",  
  "lieu": {  
    "ville": "Mons",  
    "pays": "Belgique"  
  },  
  "inscrits": [9, 0, 13, 5, 1, 14]  
}
```

"titre" \mapsto chaîne de caractère

"lieu" \mapsto objet tel que

"ville" \mapsto chaîne de caractère

"pays" \mapsto chaîne de caractère

"inscrits" \mapsto tableau de 6 entiers

- ▶ Un objet est une collection **non-ordonnée** de paires clé-valeur.
- ▶ Un tableau est une collection **ordonnée** de valeurs.

On veut vérifier que le document satisfait certaines contraintes. Notre approche² :

- ▶ On apprend un automate \mathcal{A} avec un ordre sur les clés.
- ▶ On abstrait \mathcal{A} pour permettre n'importe quel ordre.

2. BRUYÈRE, PÉREZ et STAQUET, *Validating JSON Documents with Learned VPAs*, 2022.

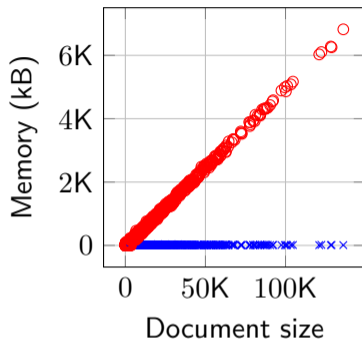
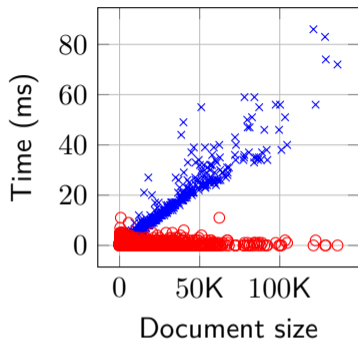




Figure 3 – Résultats expérimentaux de notre algorithme pour la vérification de documents JSON. Les croix bleues donnent les valeurs pour notre algorithme et les ronds rouges pour l'algorithme « classique ».

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).
-  BRUYÈRE, V., G. A. PÉREZ et G. STAQUET. *Validating JSON Documents with Learned VPAs*. Pre-print. Soumis à TACAS 2023. F.R.S.-FNRS, Universités de Mons et d'Anvers, 2022.