L'apprentissage d'automates et ses applications

Mois du doctorant 2021

Gaëtan Staquet
http://informatique.umons.ac.be/staff/Staquet.Gaetan/

Service d'Informatique Théorique
Département d'Informatique
Faculté des Sciences
Université de Mons

Formal Techniques in Software Engineering
Departement Informatica
Faculteit Wetenschappen
Universiteit Antwerpen

10 mars 2021

UMONS
Université de Mons

fnrs
LA LIBERTÉ DE CHERCHER

University of Antwerp

Motivation via un exemple de projet

- But du projet : implémente un jeu du pendu.

Motivation via un exemple de projet

- But du projet : implémente un jeu du pendu.
- 1re étape : créer un cahier des charges qui reprend les règles du jeu.

Motivation via un exemple de projet

- But du projet : implémente un jeu du pendu.
- 1re étape : créer un cahier des charges qui reprend les règles du jeu.
- 2e étape : implémenter le jeu.

Motivation via un exemple de projet

- But du projet : implémente un jeu du pendu.
- 1re étape : créer un cahier des charges qui reprend les règles du jeu.
- 2e étape : implémenter le jeu.
- 3e étape : vérifier l'implémentation.
  $\hookrightarrow$ Est-ce que l'implémentation répond au cahier des charges ?

Motivation via un exemple de projet

- But du projet : implémente un jeu du pendu.
- 1re étape : créer un cahier des charges qui reprend les règles du jeu.
- 2e étape : implémenter le jeu.
- 3e étape : vérifier l'implémentation.
  $\hookrightarrow$ Est-ce que l'implémentation répond au cahier des charges ?

Comment faire la 3e étape ?

Model checking

Une solution est le « model checking ».

Model checking

Une solution est le « model checking ».

- On construit une abstraction, un modèle de l'implémentation.

Model checking

Une solution est le « model checking ».

- On construit une abstraction, un modèle de l'implémentation.
- On vérifie le cahier des charges sur cette version plus simple.

Un modèle pour le jeu du pendu

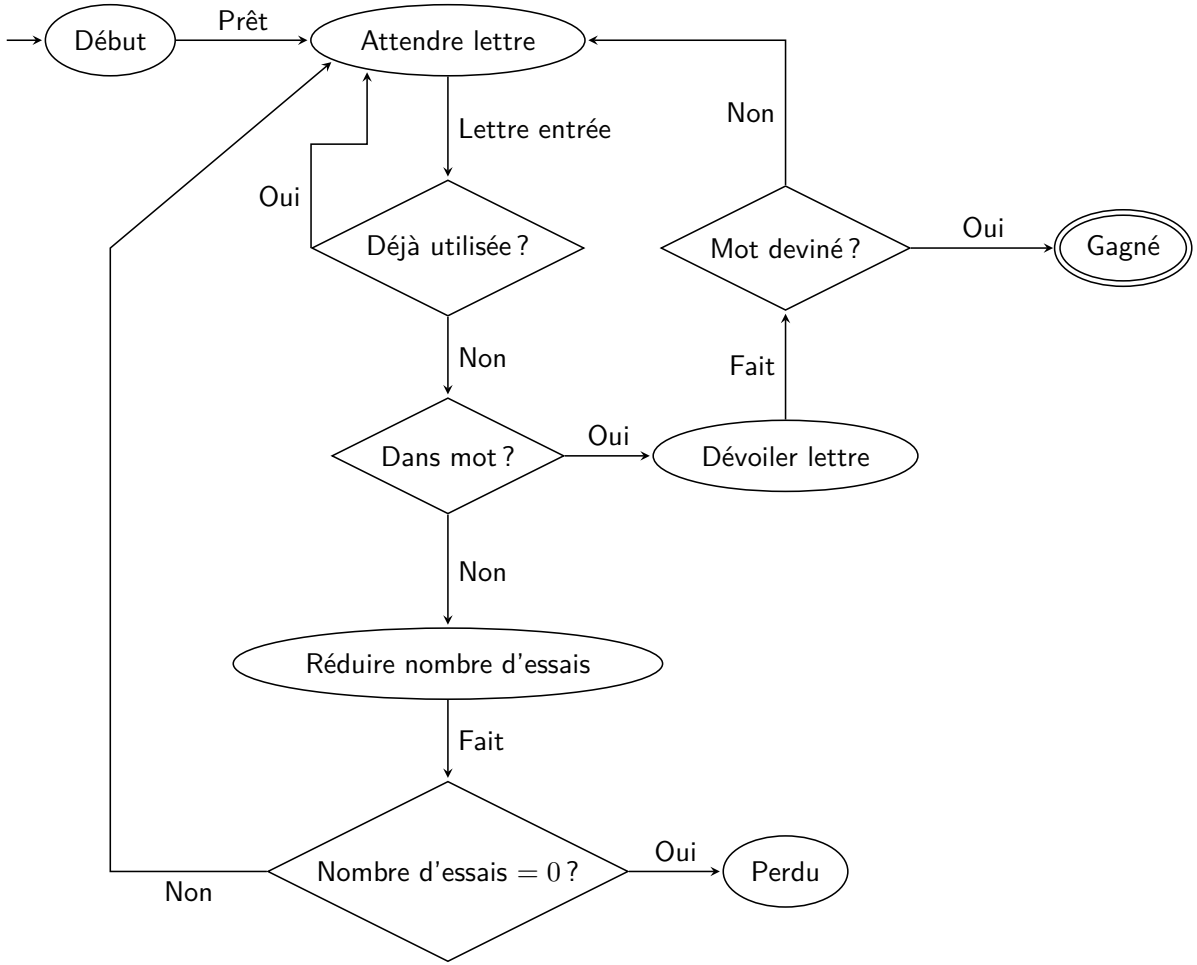


FIGURE 1 – Un modèle pour le jeu du pendu.

Simplifions le modèle

Ce modèle est encore trop compliqué. On va retirer les questions :

- On peut classer les lettres en 3 catégories :

Simplifions le modèle

Ce modèle est encore trop compliqué. On va retirer les questions :

- On peut classer les lettres en 3 catégories :
  1. Lettre déjà vue ;

Simplifions le modèle

Ce modèle est encore trop compliqué. On va retirer les questions :

- On peut classer les lettres en 3 catégories :
  1. Lettre déjà vue ;
  2. Lettre nouvelle et présente dans le mot ; ou

Simplifions le modèle

Ce modèle est encore trop compliqué. On va retirer les questions :

- On peut classer les lettres en 3 catégories :
  1. Lettre déjà vue ;
  2. Lettre nouvelle et présente dans le mot ; ou
  3. Lettre nouvelle et pas présente dans le mot.

Simplifions le modèle

Ce modèle est encore trop compliqué. On va retirer les questions :

- On peut classer les lettres en 3 catégories :
  1. Lettre déjà vue ;
  2. Lettre nouvelle et présente dans le mot ; ou
  3. Lettre nouvelle et pas présente dans le mot.
- On suppose que le système envoie directement un signal « Trouvé » ou « Pas trouvé » après avoir dévoilé la lettre dans le mot.

Simplifions le modèle

Ce modèle est encore trop compliqué. On va retirer les questions :

- On peut classer les lettres en 3 catégories :
  1. Lettre déjà vue ;
  2. Lettre nouvelle et présente dans le mot ; ou
  3. Lettre nouvelle et pas présente dans le mot.
- On suppose que le système envoie directement un signal « Trouvé » ou « Pas trouvé » après avoir dévoilé la lettre dans le mot.
- On n'est pas convaincu que l'implémentation du nombre d'erreurs soit correcte.
  $\hookrightarrow$ On veut vérifier explicitement cette partie-là. On suppose qu'on perd après deux erreurs.

Un modèle encore plus abstrait

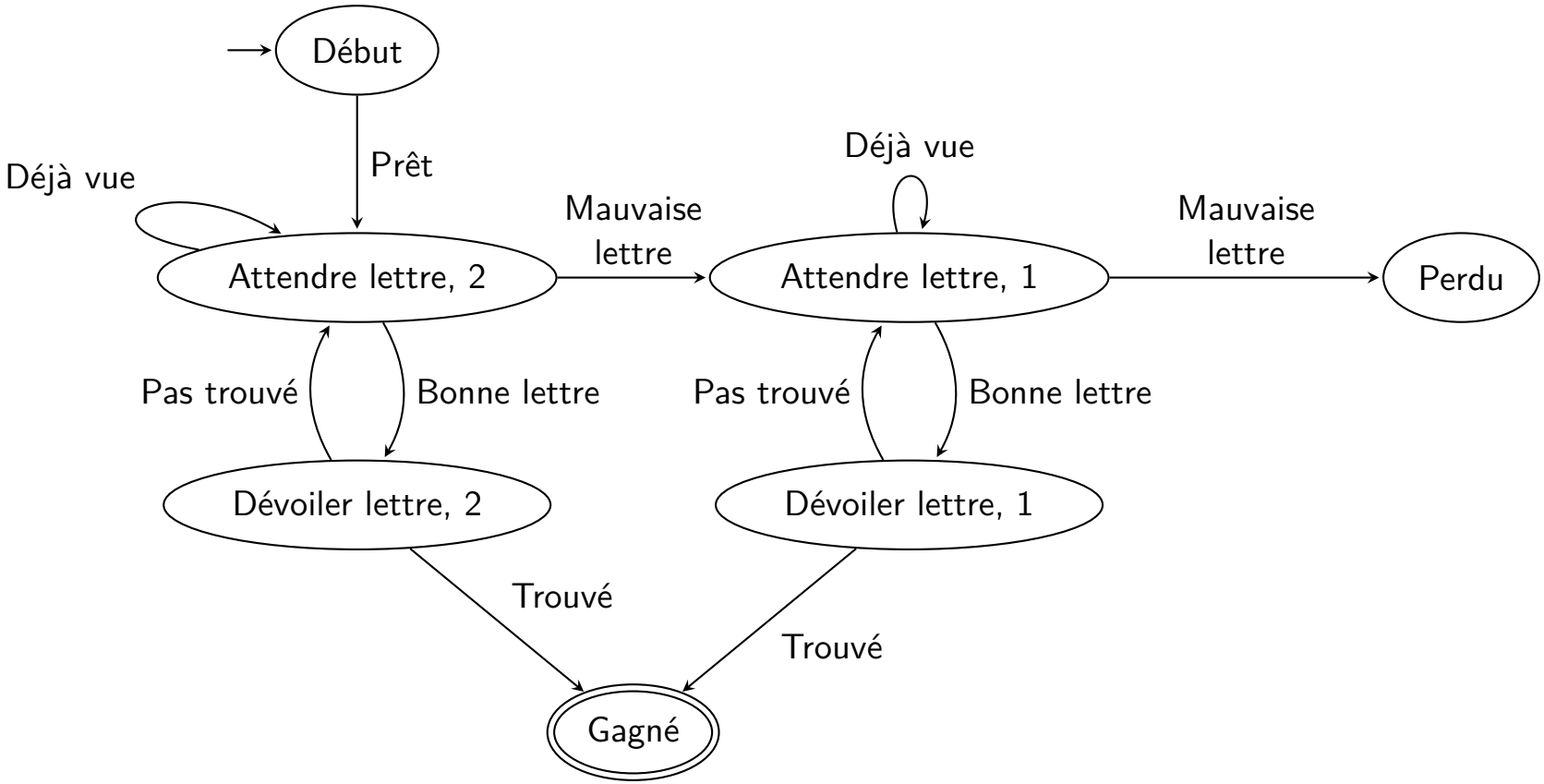


FIGURE 2 – Un autre modèle plus abstrait.

Un modèle encore plus abstrait

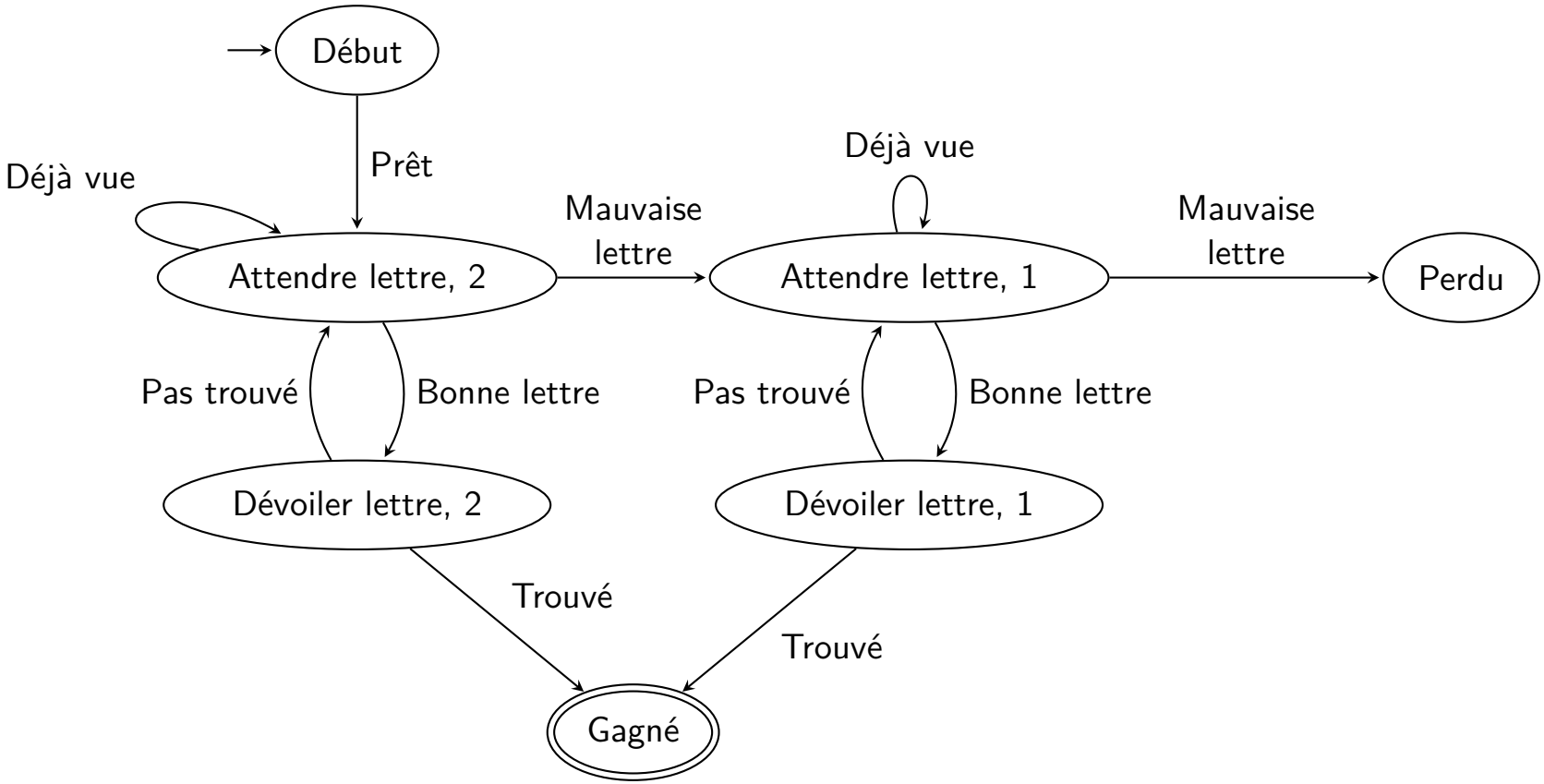


FIGURE 2 – Un autre modèle plus abstrait.

On obtient un modèle simple qui représente le système.

Un modèle encore plus abstrait

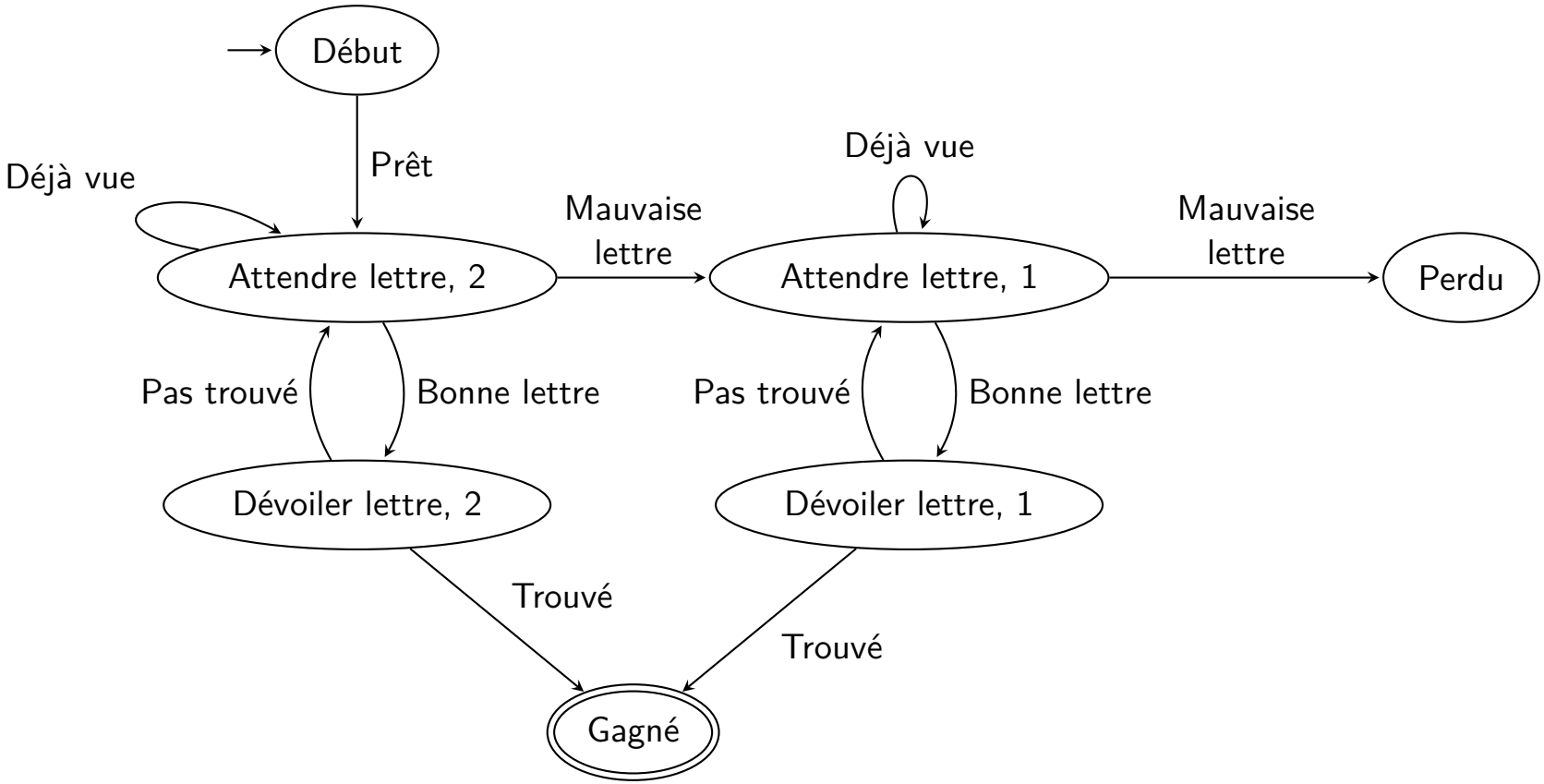


FIGURE 2 – Un autre modèle plus abstrait.

On obtient un modèle simple qui représente le système. Ce type de modèle s'appelle automate fini.

Alphabets

Définition 1 (Hopcroft et Ullman 1979)

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

Exemple 2

L'ensemble $\{0, 1\}$ est l'alphabet sur les symboles binaires, tandis que $\{a, b\}$ est l'alphabet sur les lettres a et b.

Mots

Définition 3 (Hopcroft et Ullman 1979)

Soit Σ un alphabet. Un mot sur Σ est une séquence finie de symboles de Σ, i.e., $w = a_1 a_2 \dots a_n$ (avec $n \in \mathbb{N}$) est un mot si et seulement si $\forall i, a_i \in \Sigma$.

Exemple 4

La séquence $w = 00101010$ est un mot sur $\{0, 1\}$.

Mots

Définition 3 (Hopcroft et Ullman 1979)

Soit Σ un alphabet. Un mot sur Σ est une séquence finie de symboles de Σ, i.e., $w = a_1 a_2 \dots a_n$ (avec $n \in \mathbb{N}$) est un mot si et seulement si $\forall i, a_i \in \Sigma$.

Exemple 4

La séquence $w = 00101010$ est un mot sur $\{0, 1\}$.

Définition 5 (Hopcroft et Ullman 1979)

La longueur d'un mot w, notée $|w|$, est le nombre de symboles dans w, i.e., $|w| = n$ si et seulement si $w = a_1 \dots a_n$ (avec $n \in \mathbb{N}$).

Exemple 6

La longueur de w est huit, càd, $|w| = 8$.

Mots – suite

Définition 7 (Hopcroft et Ullman 1979)

Le nombre d'occurrences de $a \in \Sigma$ dans un mot w sur Σ est noté $|w|_a$.

Exemple 8

Soit $w = 00101010$, un mot sur $\{0, 1\}$. Il y a cinq 0 dans ce mot, i.e., $|w|_0 = 5$.

Mots – suite

Définition 7 (Hopcroft et Ullman 1979)

Le nombre d'occurrences de $a \in \Sigma$ dans un mot w sur Σ est noté $|w|_a$.

Exemple 8

Soit $w = 00101010$, un mot sur $\{0, 1\}$. Il y a cinq 0 dans ce mot, i.e., $|w|_0 = 5$.

Définition 9 (Hopcroft et Ullman 1979)

Le mot vide, noté ε, est l'unique mot de longueur zéro.

Langages

Définition 10 (Hopcroft et Ullman 1979)

L'ensemble de tous les mots possibles sur un alphabet Σ est dénoté par Σ^*.

Exemple 11

Si on a $\Sigma = \{0, 1\}$, alors $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \dots\}$.

Langages

Définition 10 (Hopcroft et Ullman 1979)

L'ensemble de tous les mots possibles sur un alphabet Σ est dénoté par Σ^*.

Exemple 11

Si on a $\Sigma = \{0, 1\}$, alors $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \dots\}$.

Définition 12 (Hopcroft et Ullman 1979)

Un langage L sur Σ est un ensemble de mots sur Σ, i.e., $L \subseteq \Sigma^*$.

Exemple 13

Les ensembles $L_1 = \emptyset, L_2 = \{0, 00\}$ et $L_3 = \{w \in \Sigma^* \mid |w| = 2\}$ sont trois langages sur Σ.

Langages

Définition 10 (Hopcroft et Ullman 1979)

L'ensemble de tous les mots possibles sur un alphabet Σ est dénoté par Σ^*.

Exemple 11

Si on a $\Sigma = \{0, 1\}$, alors $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \dots\}$.

Définition 12 (Hopcroft et Ullman 1979)

Un langage L sur Σ est un ensemble de mots sur Σ, i.e., $L \subseteq \Sigma^*$.

Exemple 13

Les ensembles $L_1 = \emptyset, L_2 = \{0, 00\}$ et $L_3 = \{w \in \Sigma^* \mid |w| = 2\}$ sont trois langages sur Σ.
L'ensemble $L_4 = \{w \in \{a, b\}^* \mid |w|_a$ est pair et $|w|_b$ est impair$\}$ est un langage sur $\{a, b\}$.

Automates finis

Définition 14 (Hopcroft et Ullman 1979)

Un automate fini est un 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ où

- Q est l'ensemble des états ;
- Σ est l'alphabet ;
- $\delta : Q \times \Sigma \to Q$ est la fonction de transition ;
- q_0 est l'état initial ; et
- F est l'ensemble des états acceptants.

Automates finis – Exemples

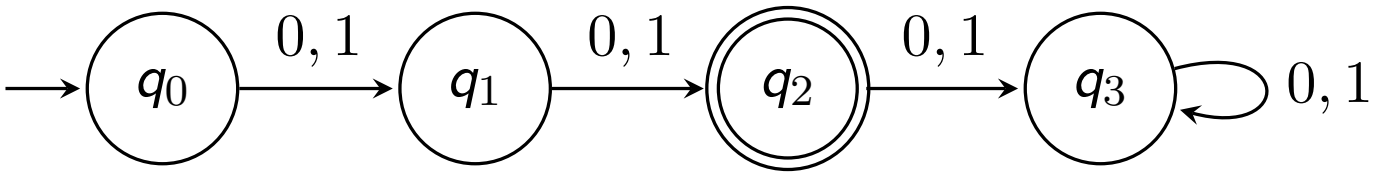


FIGURE 3 – Un automate fini.

Automates finis – Exemples

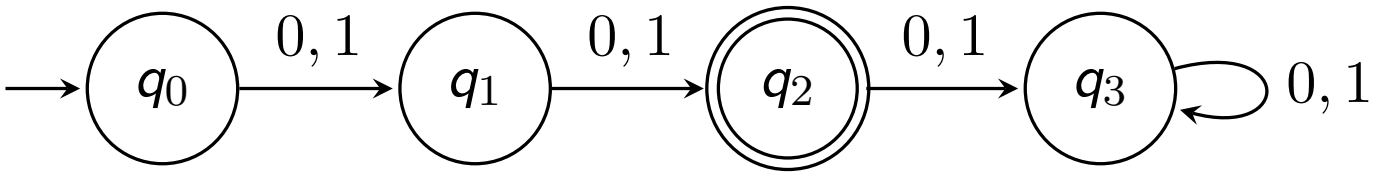


FIGURE 3 – Un automate fini acceptant $L_3 = \{w \in \{0,1\}^* \mid |w| = 2\}$.

Automates finis – Exemples

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

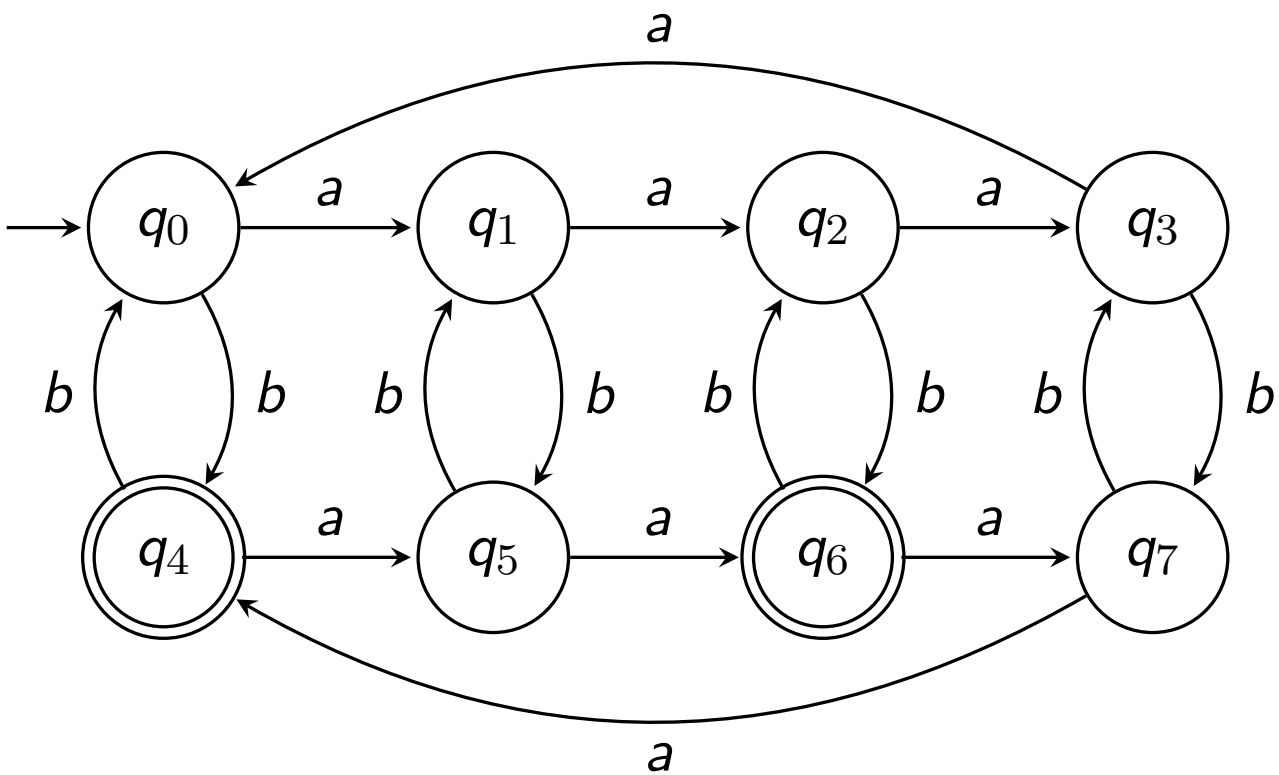


FIGURE 4 – Automate fini acceptant L_4.

Chemins et langage d'un automate

Définition 15 (Hopcroft et Ullman 1979)

Soient Σ un alphabet et $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un automate.
Pour tout mot $w = a_1 \dots a_n$ (avec $n \in \mathbb{N}$) sur Σ, on définit le chemin de w dans $\mathcal{A}$ comme :

$$p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \xrightarrow{a_3} \dots \xrightarrow{a_n} p_n$$

avec $p_0 = q_0$ et $\forall i \in \{0, \dots, n-1\}, p_i \xrightarrow{a_{i+1}} p_{i+1}$ si et seulement si $\delta(p_i, a_{i+1}) = p_{i+1}$.

Chemins et langage d'un automate

Définition 15 (Hopcroft et Ullman 1979)

Soient Σ un alphabet et $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un automate.
Pour tout mot $w = a_1 \dots a_n$ (avec $n \in \mathbb{N}$) sur Σ, on définit le chemin de w dans $\mathcal{A}$ comme :

$$p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \xrightarrow{a_3} \dots \xrightarrow{a_n} p_n$$

avec $p_0 = q_0$ et $\forall i \in \{0, \dots, n-1\}, p_i \xrightarrow{a_{i+1}} p_{i+1}$ si et seulement si $\delta(p_i, a_{i+1}) = p_{i+1}$.
Si $p_n \in F$, alors le chemin est dit acceptant et w est accepté par $\mathcal{A}$.

Chemins et langage d'un automate

Définition 15 (Hopcroft et Ullman 1979)

Soient Σ un alphabet et $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un automate.
Pour tout mot $w = a_1 \dots a_n$ (avec $n \in \mathbb{N}$) sur Σ, on définit le chemin de w dans $\mathcal{A}$ comme :

$$p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \xrightarrow{a_3} \dots \xrightarrow{a_n} p_n$$

avec $p_0 = q_0$ et $\forall i \in \{0, \dots, n-1\}, p_i \xrightarrow{a_{i+1}} p_{i+1}$ si et seulement si $\delta(p_i, a_{i+1}) = p_{i+1}$.
Si $p_n \in F$, alors le chemin est dit acceptant et w est accepté par $\mathcal{A}$.
Le langage de $\mathcal{A}$, noté $\mathcal{L}(\mathcal{A})$, est l'ensemble des mots acceptés par $\mathcal{A}$, i.e.,

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \exists q \in F, q_0 \xrightarrow{w} q\}.$$

Relation de Myhill-Nerode

Définissons une relation d'équivalence en utilisant un langage.

Définition 16

Soit L un langage sur un alphabet Σ.
À partir de L, définissons la relation de Myhill-Nerode $\sim_L$ comme suit : pour tout $u, v \in \Sigma^*$, u et v sont en relation, noté $u \sim_L v$, si et seulement si $\forall w \in \Sigma^*, uw \in L \iff vw \in L$.
On dénote la classe d'équivalence de w par $[\![w]\!]_L$.

Relation de Myhill-Nerode – Exemple

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.
Soit $w \in \{a, b\}^*$.

Relation de Myhill-Nerode – Exemple

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.
Soit $w \in \{a, b\}^*$. La relation de Myhill-Nerode contient quatre classes d'équivalence, c'est-à-dire qu'on peut ranger w dans exactement une dans quatre catégories suivantes :

Relation de Myhill-Nerode – Exemple

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.
Soit $w \in \{a, b\}^*$. La relation de Myhill-Nerode contient quatre classes d'équivalence, c'est-à-dire qu'on peut ranger w dans exactement une dans quatre catégories suivantes :

1. Le nombre de a est pair et le nombre de b est pair ;

Relation de Myhill-Nerode – Exemple

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.
Soit $w \in \{a, b\}^*$. La relation de Myhill-Nerode contient quatre classes d'équivalence, c'est-à-dire qu'on peut ranger w dans exactement une dans quatre catégories suivantes :

1. Le nombre de a est pair et le nombre de b est pair ;
2. Le nombre de a est pair et le nombre de b est impair ;

Relation de Myhill-Nerode – Exemple

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.
Soit $w \in \{a, b\}^*$. La relation de Myhill-Nerode contient quatre classes d'équivalence, c'est-à-dire qu'on peut ranger w dans exactement une dans quatre catégories suivantes :

1. Le nombre de a est pair et le nombre de b est pair ;
2. Le nombre de a est pair et le nombre de b est impair ;
3. Le nombre de a est impair et le nombre de b est pair ; et

Relation de Myhill-Nerode – Exemple

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.
Soit $w \in \{a, b\}^*$. La relation de Myhill-Nerode contient quatre classes d'équivalence, c'est-à-dire qu'on peut ranger w dans exactement une dans quatre catégories suivantes :

1. Le nombre de a est pair et le nombre de b est pair ;
2. Le nombre de a est pair et le nombre de b est impair ;
3. Le nombre de a est impair et le nombre de b est pair ; et
4. Le nombre de a est impair et le nombre de b est impair.

De le relation de Myhill-Nerode vers un automate

Une fois toutes les classes d'équivalence de la relation de Myhill-Nerode pour un langage L identifiées, on peut construire un automate $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ acceptant L avec :

- $Q = \{[\![w]\!]_L \mid w \in \Sigma^*\}$;
- $\forall w \in \Sigma^*, \forall a \in \Sigma, \delta([\![w]\!]_L, a) = [\![wa]\!]_L$;
- $q_0 = [\![\varepsilon]\!]_L$; et
- $F = \{[\![w]\!]_L \mid w \in L\}$.

De le relation de Myhill-Nerode vers un automate

Une fois toutes les classes d'équivalence de la relation de Myhill-Nerode pour un langage L identifiées, on peut construire un automate $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ acceptant L avec :

- $Q = \{[\![w]\!]_L \mid w \in \Sigma^*\}$;
- $\forall w \in \Sigma^*, \forall a \in \Sigma, \delta([\![w]\!]_L, a) = [\![wa]\!]_L$;
- $q_0 = [\![\varepsilon]\!]_L$; et
- $F = \{[\![w]\!]_L \mid w \in L\}$.

Lemme 17 (Hopcroft et Ullman 1979)

Soit L.
Il existe un automate fini $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = L$ si et seulement si le nombre de classes d'équivalence de $\sim_L$ est fini.
On appelle un tel langage régulier.

Un exemple de construction

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

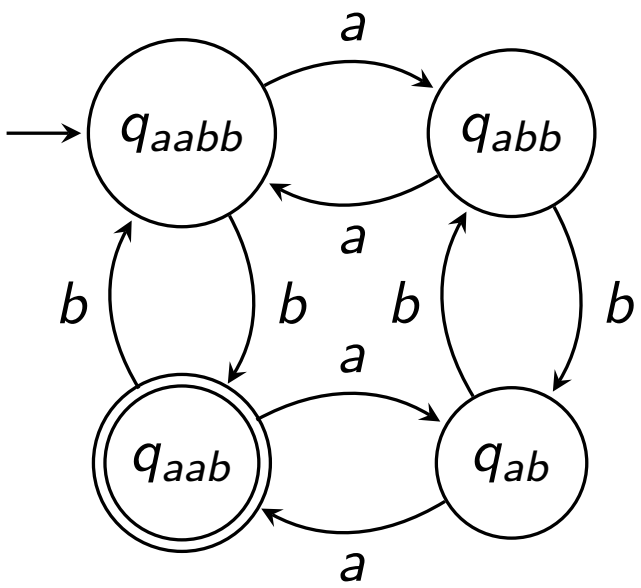


FIGURE 5 – Automate acceptant L_4 construit depuis $\sim_{L_4}$.

Apprendre un automate fini

Soit un alphabet Σ. On veut un algorithme qui prend en entrée un langage L sur Σ et retourne un automate fini.

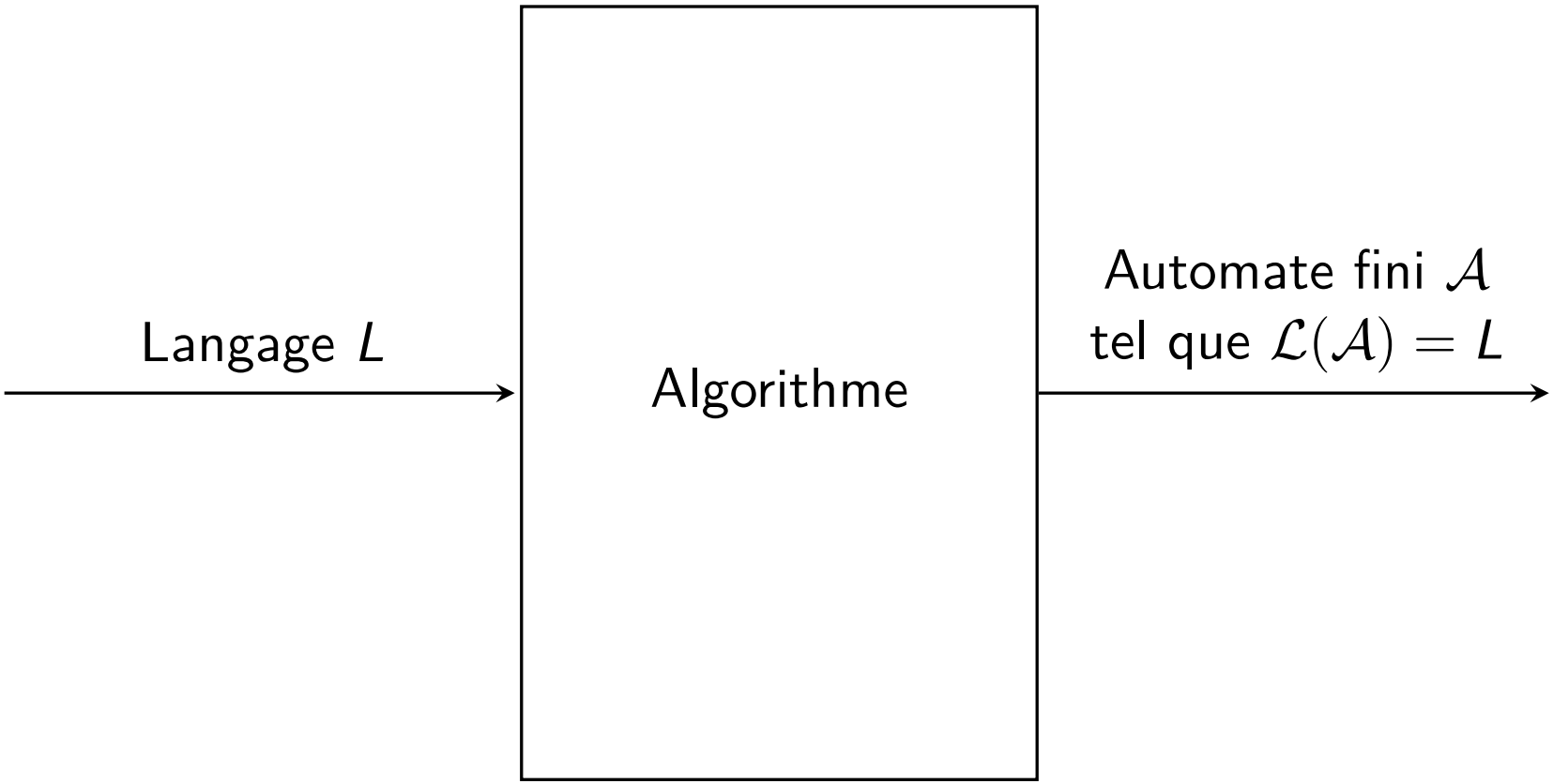


FIGURE 6 – Représentation de l'algorithme.

Apprendre un automate fini

Soit un alphabet Σ. On veut un algorithme qui prend en entrée un langage L sur Σ et retourne un automate fini.

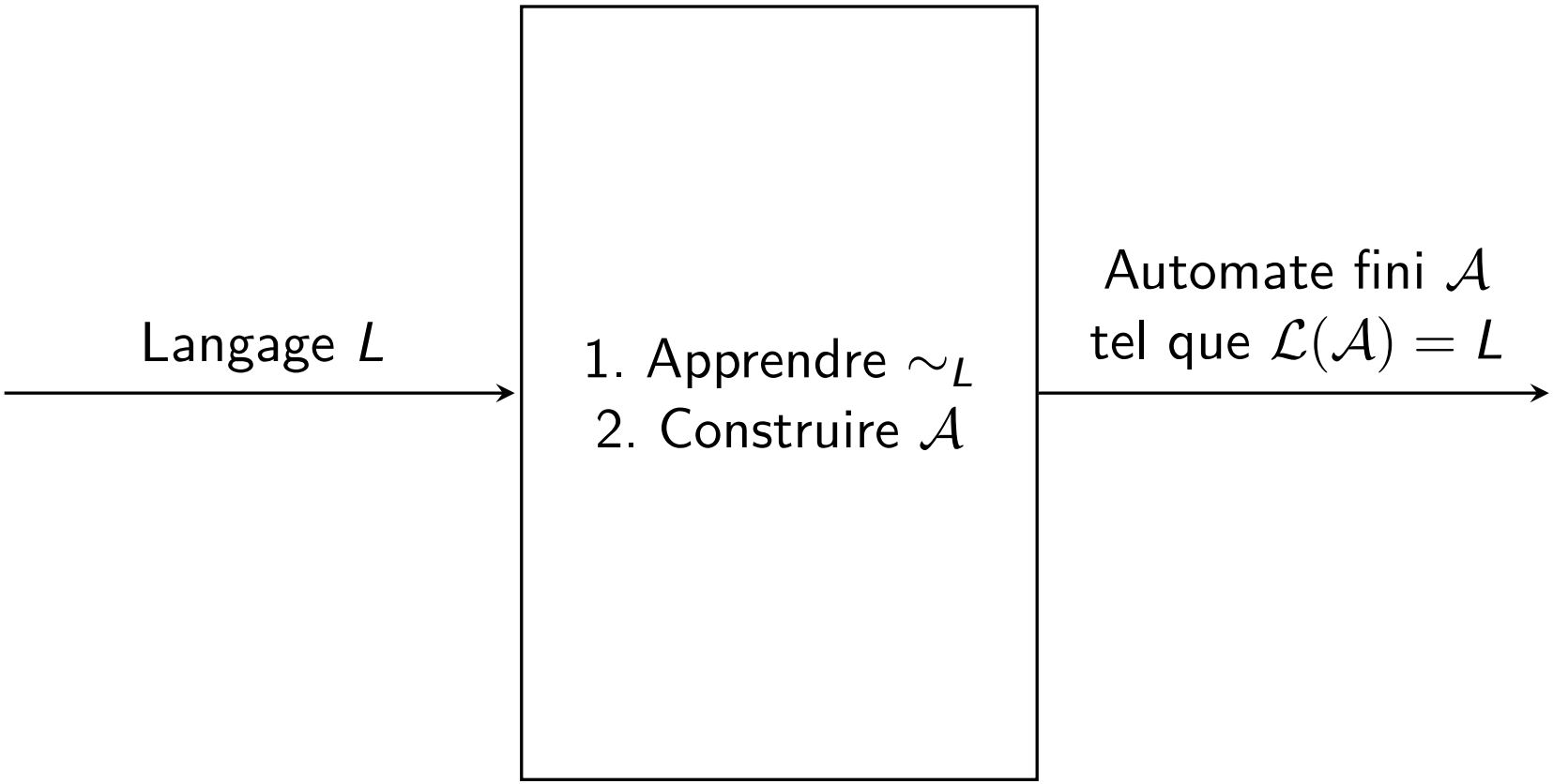


FIGURE 6 – Représentation de l'algorithme.

Framework d'Angluin (Angluin 1987)

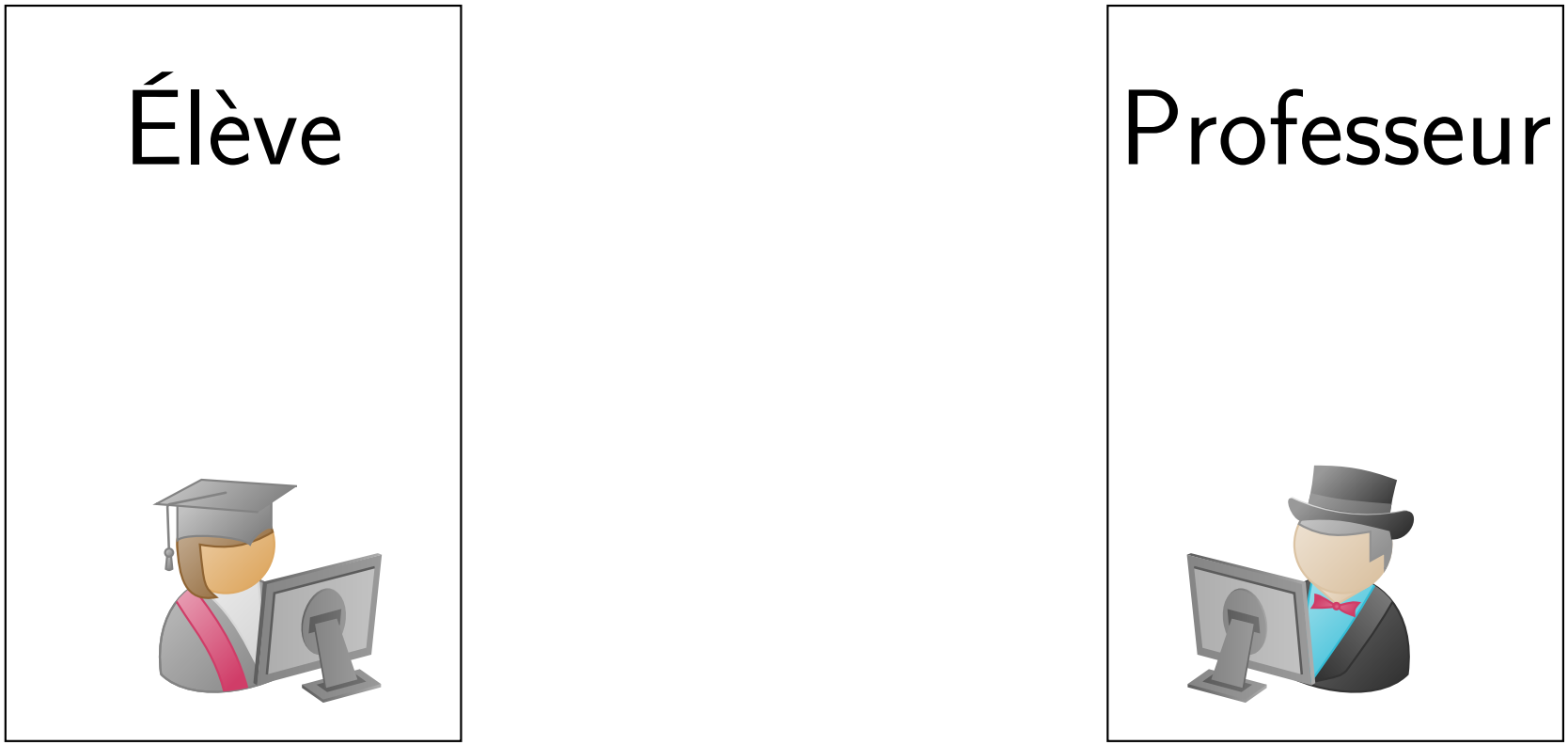


Figure 7 – Le modèle élève-professeur (Angluin 1987).

Framework d'Angluin (Angluin 1987)

Élève

Va identifier les classes d'équivalence de $\sim_L$ et construire un automate

Professeur

Connaît un automate acceptant L

Figure 7 – Le modèle élève-professeur (Angluin 1987).

Framework d'Angluin (Angluin 1987)

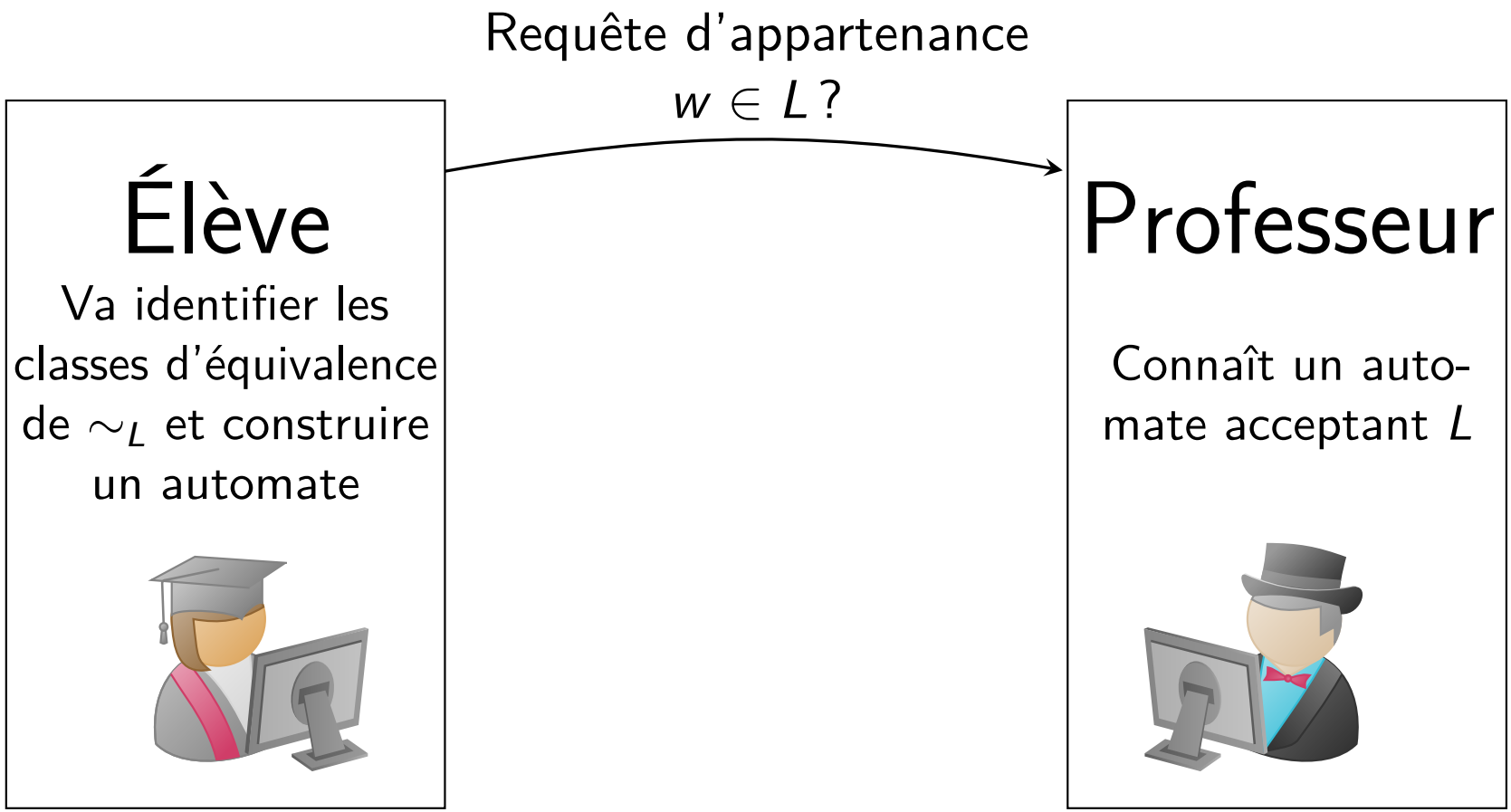


Figure 7 – Le modèle élève-professeur (Angluin 1987).

Framework d'Angluin (Angluin 1987)

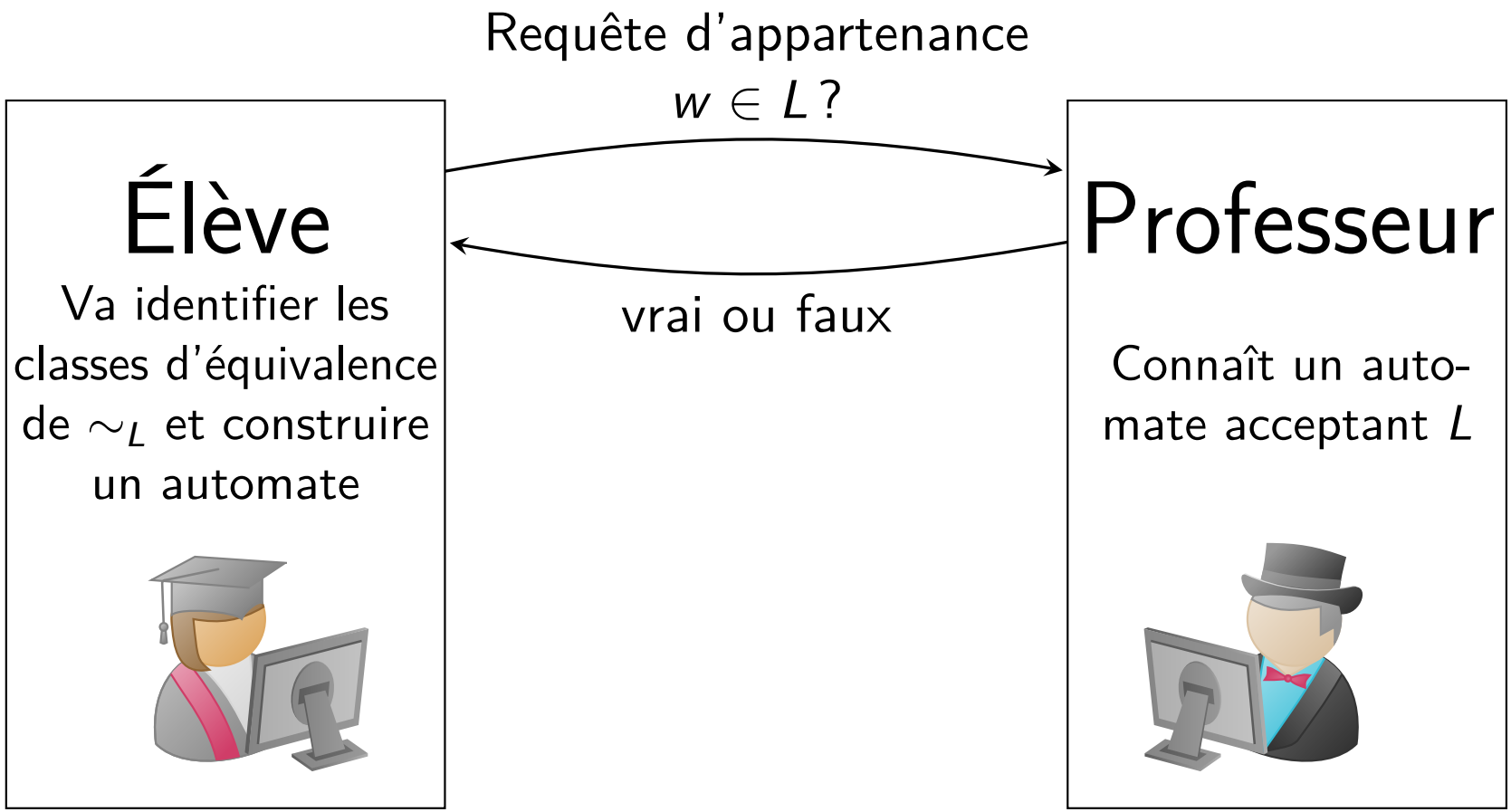


Figure 7 – Le modèle élève-professeur (Angluin 1987).

Framework d'Angluin (Angluin 1987)

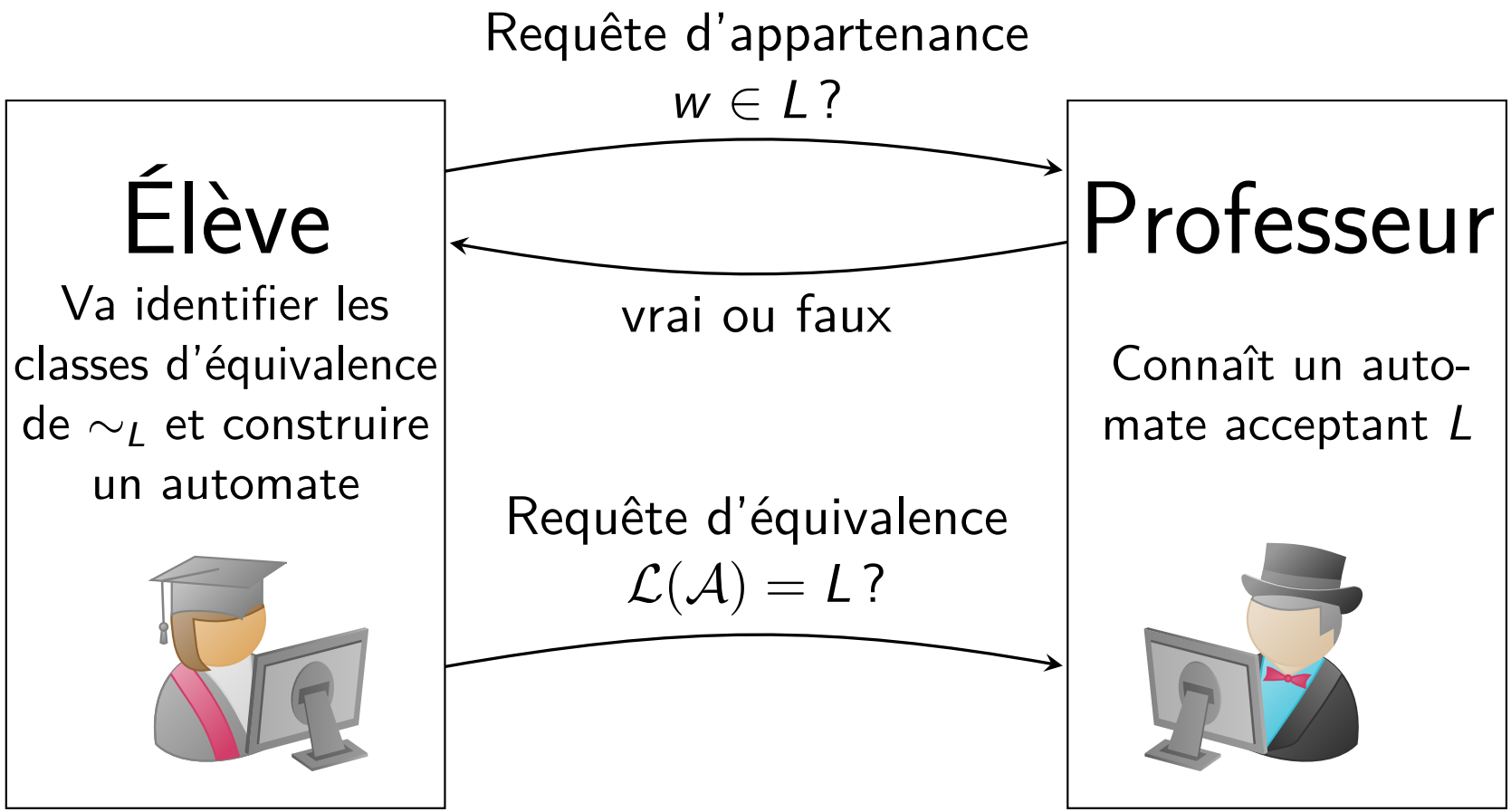


Figure 7 – Le modèle élève-professeur (Angluin 1987).

Framework d'Angluin (Angluin 1987)

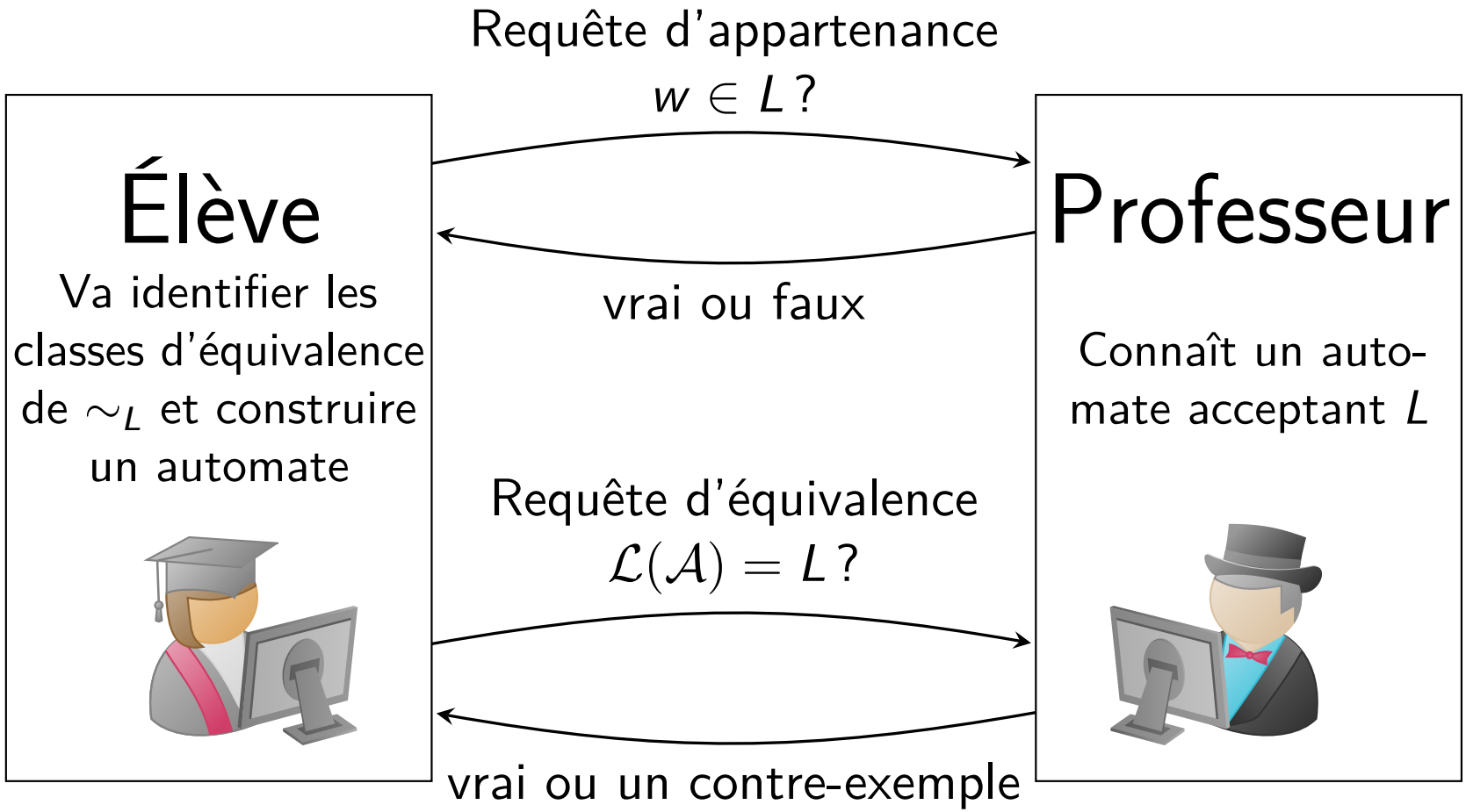


Figure 7 – Le modèle élève-professeur (Angluin 1987).

Élèves

Il existe plusieurs algorithmes pour l'élève. Le transparent suivant donne l'intuition derrière l'élève L^* (Angluin 1987) sur un exemple.

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

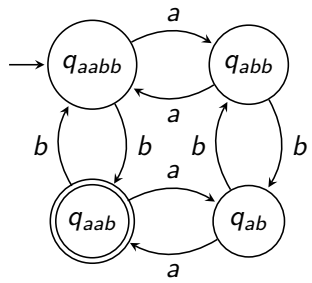


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

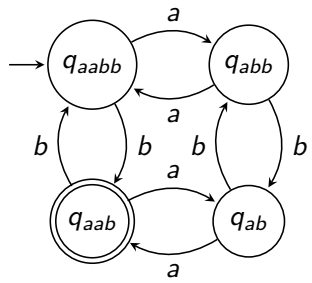
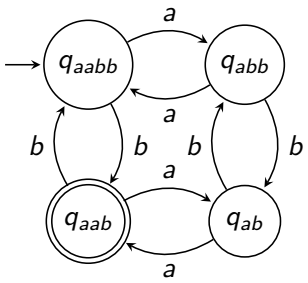


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

	ε
ε	X
a	X
b	V

Pour chaque ligne r et chaque colonne c, l'élève pose une requête d'appartenance sur $r \cdot c$. On met un V si et seulement si $r \cdot c \in L$.

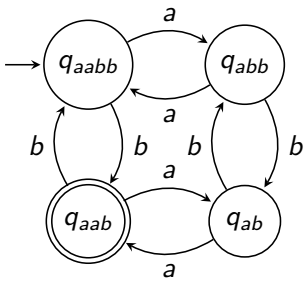
Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.



	ε
ε	X
a	X
b	V

FIGURE 8 – Un automate acceptant L_4 connu du professeur.

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.



	ε
ε	X
a	X
b	V

FIGURE 8 – Un automate acceptant L_4 connu du professeur.

b n'a pas de représentant.

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

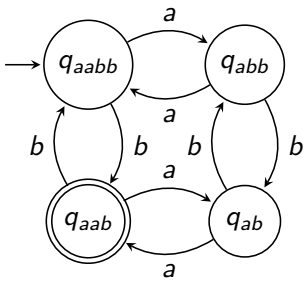


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

	ε
ε	X
b	V
a	X
ba	X
bb	X

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

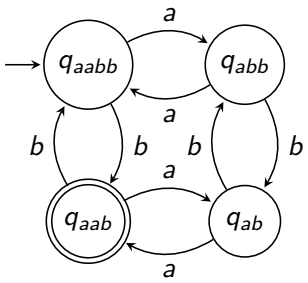


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

	ε
ε	X
b	V
a	X
ba	X
bb	X

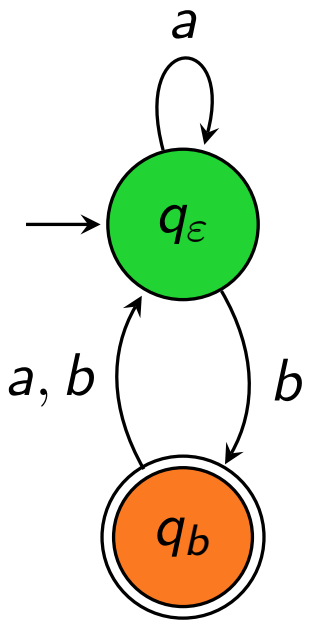


FIGURE 9 – L'automate décrit par la table.

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

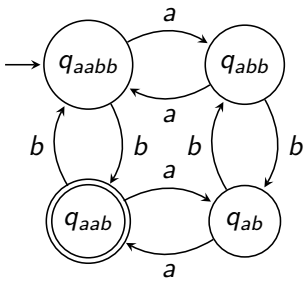


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

	ε
ε	X
b	V
a	X
ba	X
bb	X

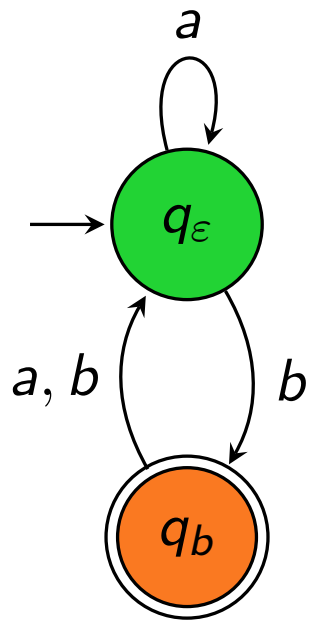


FIGURE 9 – L'automate décrit par la table.

Une fois l'automate construit, l'élève pose une requête d'équivalence. Supposons que le professeur donne le contre-exemple ab.

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

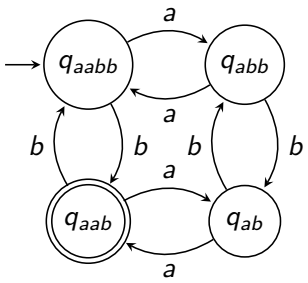


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

	ε
ε	X
b	V
a	X
ab	X
ba	X
bb	X
aa	X
aba	V
abb	X

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

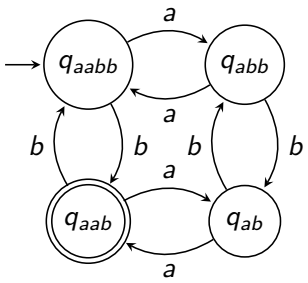


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

	ε
ε	X
b	V
a	X
ab	X
ba	X
bb	X
aa	X
aba	V
abb	X

On a ε et a mais b et ab.

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

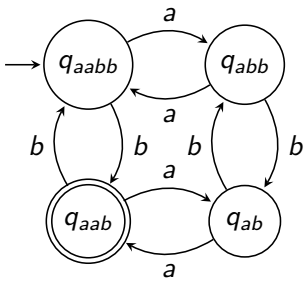


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

	ε
ε	X
b	V
a	X
ab	X
ba	X
bb	X
aa	X
aba	V
abb	X

On a ε et a mais b et ab. Donc, b sépare deux classes d'équivalence.

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

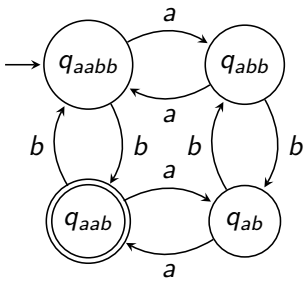


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

	ε	b
ε	X	V
b	V	X
a	X	X
ab	X	X
ba	X	X
bb	X	V
aa	X	V
aba	V	X
abb	X	X

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

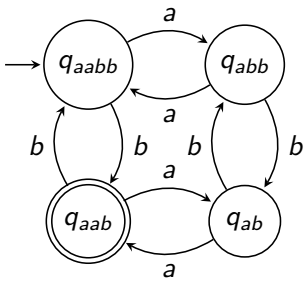


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

	ε	b
ε	X	V
b	V	X
a	X	X
ab	X	X
ba	X	X
bb	X	V
aa	X	V
aba	V	X
abb	X	X

On a a et ab mais aa et aba.

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

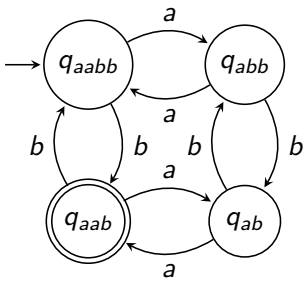


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

	ε	b
ε	X	V
b	V	X
a	X	X
ab	X	X
ba	X	X
bb	X	V
aa	X	V
aba	V	X
abb	X	X

On a a et ab mais aa et aba. Donc, a sépare deux classes d'équivalence.

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

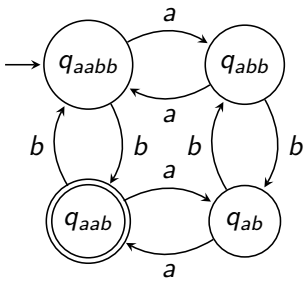


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

	ε	b	a
ε	X	V	X
b	V	X	X
a	X	X	X
ab	X	X	V
ba	X	X	V
bb	X	V	X
aa	X	V	X
aba	V	X	X
abb	X	X	X

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

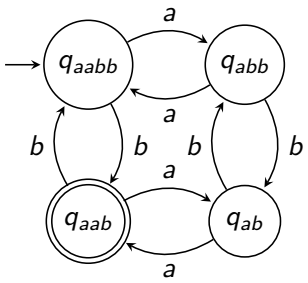


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

	ε	b	a
ε	X	V	X
b	V	X	X
a	X	X	X
ab	X	X	V
ba	X	X	V
bb	X	V	X
aa	X	V	X
aba	V	X	X
abb	X	X	X

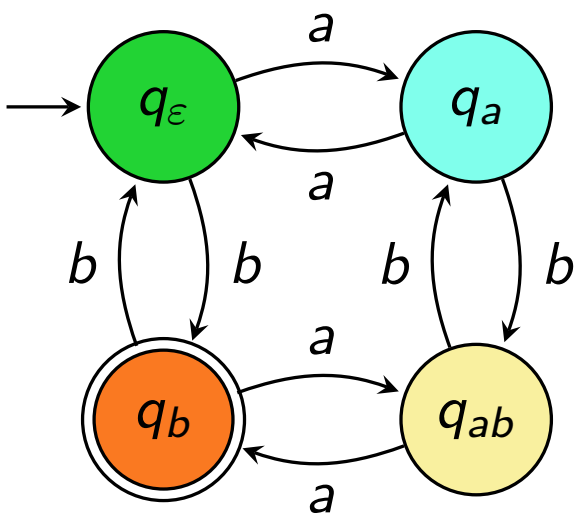


FIGURE 9 – L'automate décrit par la table.

Prenons $L_4 = \{w \in \{a, b\}^* \mid |w|_a \text{ est pair et } |w|_b \text{ est impair}\}$.

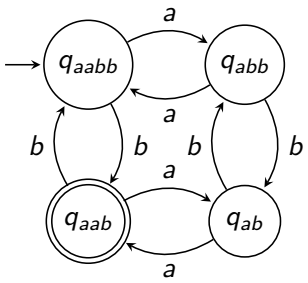


FIGURE 8 – Un automate acceptant L_4 connu du professeur.

	ε	b	a
ε	X	V	X
b	V	X	X
a	X	X	X
ab	X	X	V
ba	X	X	V
bb	X	V	X
aa	X	V	X
aba	V	X	X
abb	X	X	X

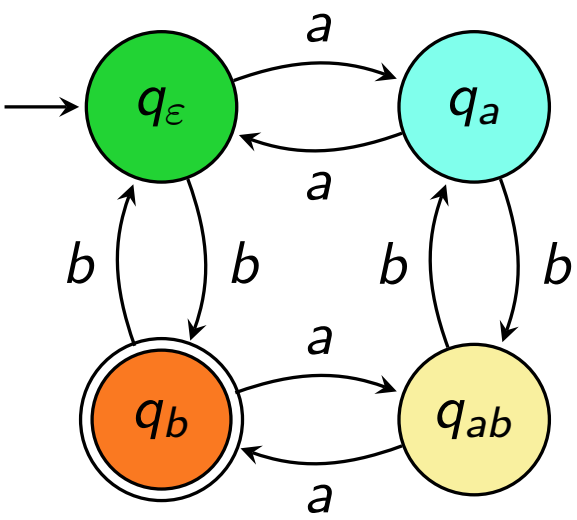


FIGURE 9 – L'automate décrit par la table.

Suite à la requête d'équivalence, le professeur retourne « vrai ». L'élève a donc fini son travail.

Retour du pendu

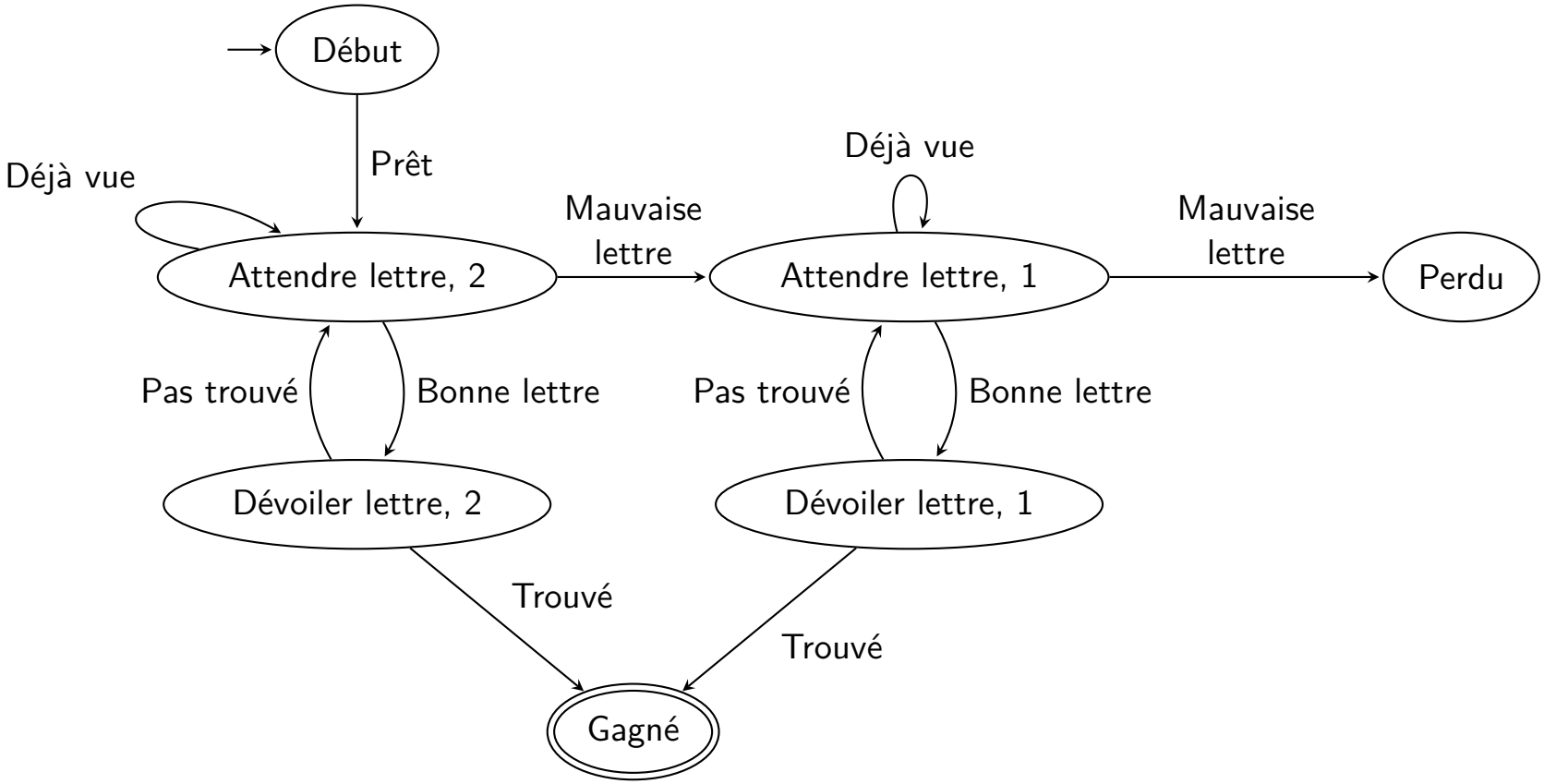


FIGURE 10 – L'automate fini construit précédemment pour le jeu du pendu.

Retour du pendu

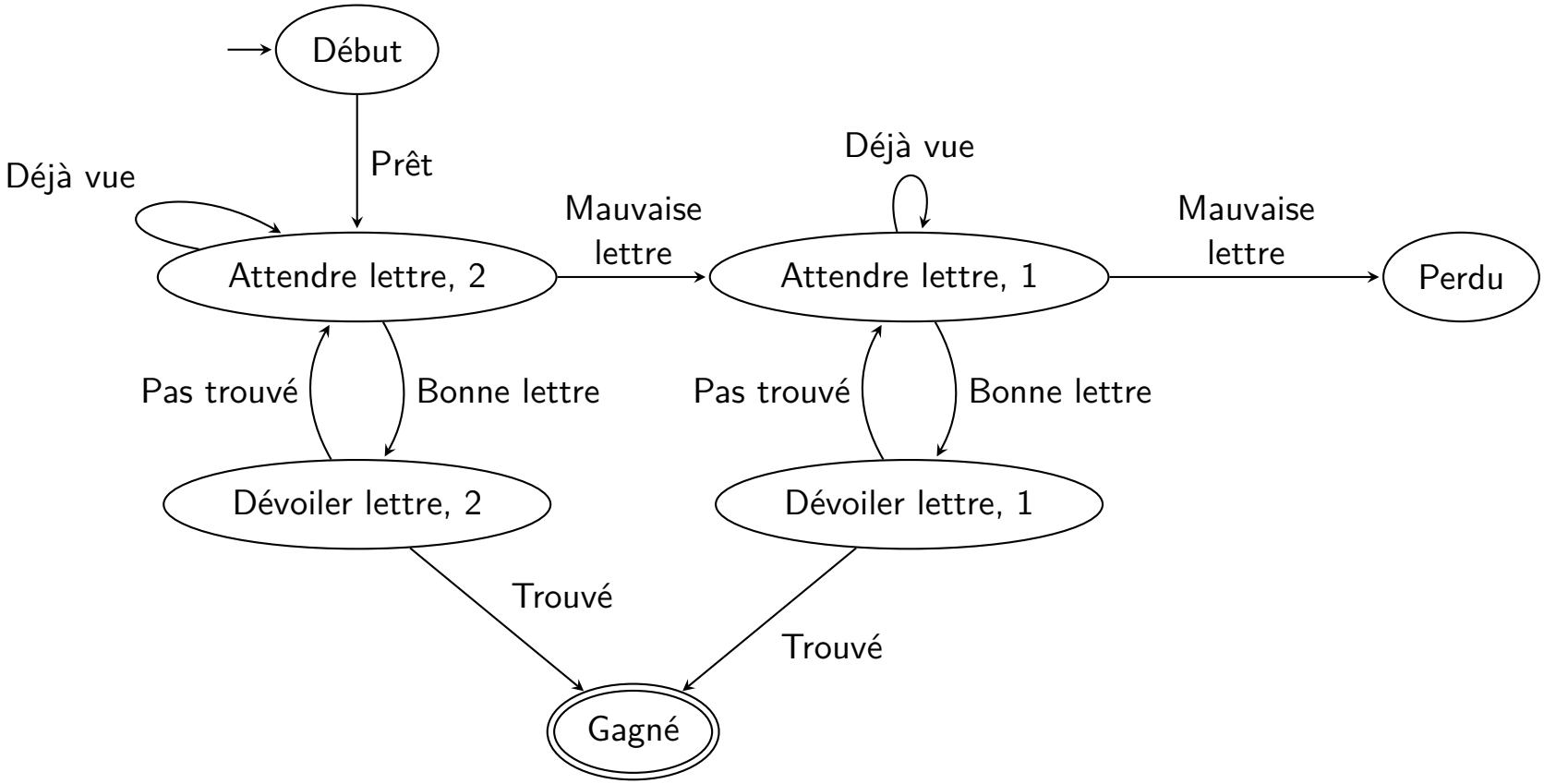


FIGURE 10 – L'automate fini construit précédemment pour le jeu du pendu.

On veut pouvoir supporter un nombre arbitraire d'erreurs.

Un nombre arbitraire d'erreurs

Si on a un nombre arbitraire d'erreurs, on ne peut pas borner le nombre maximal d'erreurs.

Un nombre arbitraire d'erreurs

Si on a un nombre arbitraire d'erreurs, on ne peut pas borner le nombre maximal d'erreurs.
$\hookrightarrow$ Le nombre de classes d'équivalence de la relation de Myhill-Nerode est infini.

Un nombre arbitraire d'erreurs

Si on a un nombre arbitraire d'erreurs, on ne peut pas borner le nombre maximal d'erreurs.
$\hookrightarrow$ Le nombre de classes d'équivalence de la relation de Myhill-Nerode est infini.
$\hookrightarrow$ Les automates finis ne suffisent pas ☹.

Retour du pendu

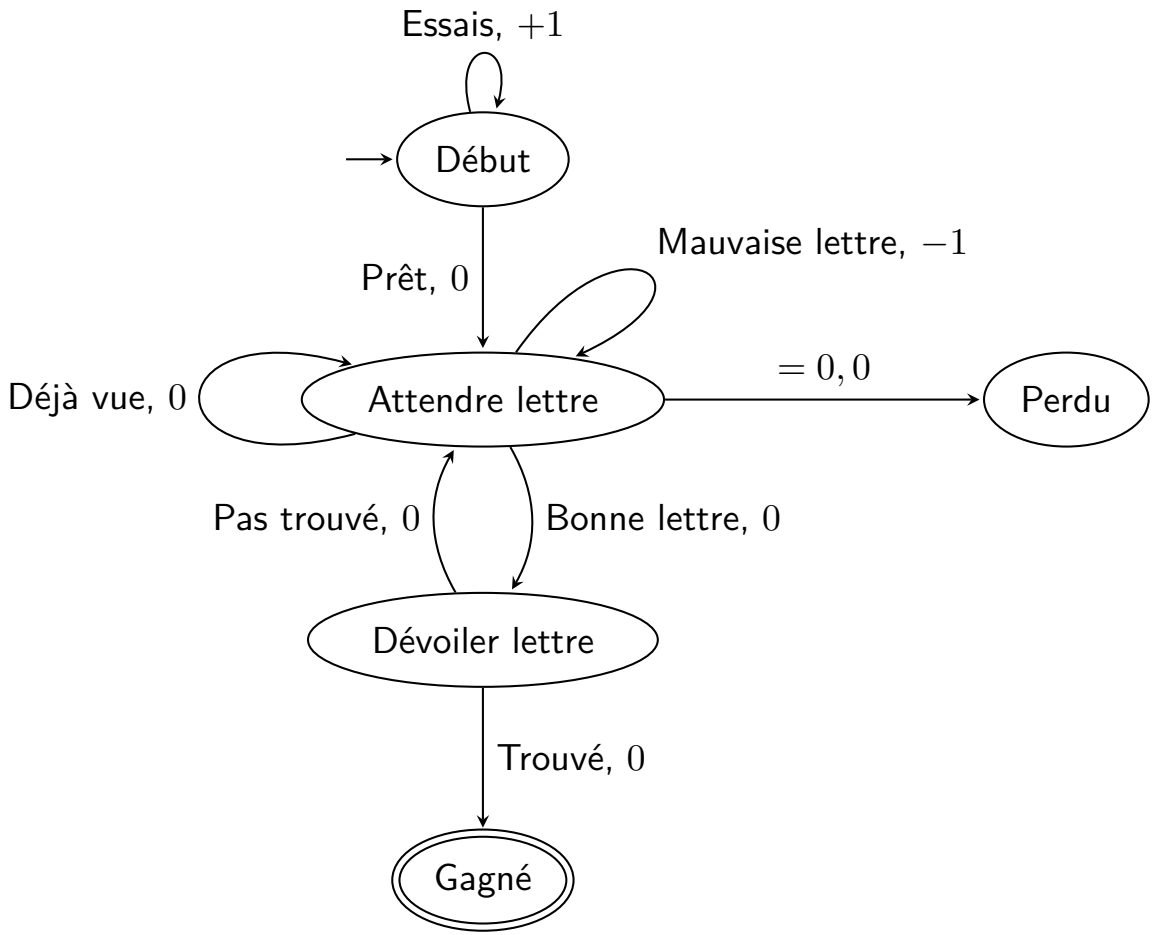


FIGURE 11 – Un automate à un compteur modélisant le jeu du pendu.

Comment apprendre un automate à un compteur ?

Comment apprendre un automate à un compteur ?

C'est une bonne question...

Comment apprendre un automate à un compteur ?

C'est une bonne question...
Rendez-vous d'ici quatre ans !

Références I


Angluin, Dana (1987). « Learning Regular Sets from Queries and Counterexamples ». In : *Inf. Comput.* 75.2, p. 87-106. DOI : 10.1016/0890-5401(87)90052-6. URL : https://doi.org/10.1016/0890-5401(87)90052-6.


Hopcroft, John E. et Jeffrey D. Ullman (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley. ISBN : 0-201-02988-X.