Apprentissage Actif avec des Contres Exemples Inductifs

Juliette Jacquot

14 janvier 2025

Sommaire

Introduction

Sommaire

Introduction

Apprentissage actif :

Introduction

Apprentissage actif :

- But : apprendre un automate à l'aide de requêtes

Introduction

Apprentissage actif :

- But : apprendre un automate à l'aide de requêtes
- Apprenant : apprend le langage, envoie les requêtes

Introduction

Apprentissage actif :

- But : apprendre un automate à l'aide de requêtes
- Apprenant : apprend le langage, envoie les requêtes
- Oracle : connaît le langage, répond aux requêtes

Introduction

Apprentissage actif :

- But : apprendre un automate à l'aide de requêtes
- Apprenant : apprend le langage, envoie les requêtes
- Oracle : connaît le langage, répond aux requêtes
- 2 types de requêtes :

Introduction

Apprentissage actif :

- But : apprendre un automate à l'aide de requêtes
- Apprenant : apprend le langage, envoie les requêtes
- Oracle : connaît le langage, répond aux requêtes
- 2 types de requêtes :
  - Appartenance : si un mot est dans le langage

Introduction

Apprentissage actif :

- But : apprendre un automate à l'aide de requêtes
- Apprenant : apprend le langage, envoie les requêtes
- Oracle : connaît le langage, répond aux requêtes
- 2 types de requêtes :
  - Appartenance : si un mot est dans le langage
  - Équivalence : si un automate est équivalent à celui cherché

Introduction

Apprentissage actif :

- But : apprendre un automate à l'aide de requêtes
- Apprenant : apprend le langage, envoie les requêtes
- Oracle : connaît le langage, répond aux requêtes
- 2 types de requêtes :
  - Appartenance : si un mot est dans le langage
  - Équivalence : si un automate est équivalent à celui cherché
- Théorie : chaque mot a une appartenance connue

Introduction

Apprentissage actif :

- But : apprendre un automate à l'aide de requêtes
- Apprenant : apprend le langage, envoie les requêtes
- Oracle : connaît le langage, répond aux requêtes
- 2 types de requêtes :
  - Appartenance : si un mot est dans le langage
  - Équivalence : si un automate est équivalent à celui cherché
- Théorie : chaque mot a une appartenance connue
- Pratique : mémoire finie impossible

Introduction

Apprentissage actif :

- But : apprendre un automate à l'aide de requêtes
- Apprenant : apprend le langage, envoie les requêtes
- Oracle : connaît le langage, répond aux requêtes
- 2 types de requêtes :
  - Appartenance : si un mot est dans le langage
  - Équivalence : si un automate est équivalent à celui cherché
- Théorie : chaque mot a une appartenance connue
- Pratique : mémoire finie impossible $\longrightarrow$ appartenances inconnues

Introduction

Projet : Création du framework L_{ICE}

Introduction

Projet : Création du framework L_{ICE}

- Basé sur l'algorithme L^* [1]

Introduction

Projet : Création du framework L_{ICE}

- Basé sur l'algorithme L^* [1]
- Table d'observation à trous

Introduction

Projet : Création du framework L_{ICE}

- Basé sur l'algorithme L^* [1]
- Table d'observation à trous $\longrightarrow$ ajout de clauses SAT [4]

Introduction

Projet : Création du framework L_{ICE}

- Basé sur l'algorithme L^* [1]
- Table d'observation à trous $\longrightarrow$ ajout de clauses SAT [4]
- Ajout de relations entre les mots :

Introduction

Projet : Création du framework L_{ICE}

- Basé sur l'algorithme L^* [1]
- Table d'observation à trous $\longrightarrow$ ajout de clauses SAT [4]
- Ajout de relations entre les mots : $w_1 \in L \Rightarrow w_2 \in L$

Contexte

Sommaire

Model Checking [2]

Model Checking [2]

- But : Vérifier si le modèle d'un système satisfait une propriété

Model Checking [2]

- But : Vérifier si le modèle d'un système satisfait une propriété
- Exemple : terminaison d'un programme

Model Checking [2]

- But : Vérifier si le modèle d'un système satisfait une propriété
- Exemple : terminaison d'un programme

Regular Model Checking [2]

Model Checking [2]

- But : Vérifier si le modèle d'un système satisfait une propriété
- Exemple : terminaison d'un programme

Regular Model Checking [2]

- États représentés par des mots

Model Checking [2]

- But : Vérifier si le modèle d'un système satisfait une propriété
- Exemple : terminaison d'un programme

Regular Model Checking [2]

- États représentés par des mots
- Transitions entre mots conservent la taille

Model Checking [2]

- But : Vérifier si le modèle d'un système satisfait une propriété
- Exemple : terminaison d'un programme

Regular Model Checking [2]

- États représentés par des mots
- Transitions entre mots conservent la taille
- Transitions représentables par transducteurs

Ensembles d'états Init et Bad

Ensembles d'états Init et Bad $\longrightarrow$ Langages Init et Bad

Contexte

Regular Model Checking

Ensembles d'états Init et Bad $\longrightarrow$ Langages Init et Bad
Transition entre états

Contexte

Regular Model Checking

Ensembles d'états Init et Bad	$\longrightarrow$	Langages Init et Bad
Transition entre états	$\longrightarrow$	Transducteur Post

Contexte

Regular Model Checking

Ensembles d'états Init et Bad	$\longrightarrow$	Langages Init et Bad
Transition entre états	$\longrightarrow$	Transducteur Post

But : Apprendre un langage L tel que :

Contexte

Regular Model Checking

Ensembles d'états Init et Bad $\longrightarrow$ Langages Init et Bad
Transition entre états $\longrightarrow$ Transducteur Post

But : Apprendre un langage L tel que :

- $\mathrm{Init} \subseteq L$

Contexte

Regular Model Checking

Ensembles d'états Init et Bad $\longrightarrow$ Langages Init et Bad
Transition entre états $\longrightarrow$ Transducteur Post

But : Apprendre un langage L tel que :

- $\mathrm{Init} \subseteq L$
- $L \cap \mathrm{Bad} = \emptyset$

Contexte

Regular Model Checking

Ensembles d'états Init et Bad $\longrightarrow$ Langages Init et Bad
Transition entre états $\longrightarrow$ Transducteur Post

But : Apprendre un langage L tel que :

- $\mathrm{Init} \subseteq L$
- $L \cap \mathrm{Bad} = \emptyset$
- $\mathrm{Post}(L) \subseteq L$

Contexte

Regular Model Checking

Ensembles d'états Init et Bad $\longrightarrow$ Langages Init et Bad
Transition entre états $\longrightarrow$ Transducteur Post

But : Apprendre un langage L tel que :

- $\mathrm{Init} \subseteq L$
- $L \cap \mathrm{Bad} = \emptyset$
- $\mathrm{Post}(L) \subseteq L$

Habituellement : $L = \mathrm{Post}^*(\mathrm{Init})$ ou $L = \Sigma^* \setminus \mathrm{Pre}^*(\mathrm{Bad})$

Contexte

Regular Model Checking

Ensembles d'états Init et Bad $\longrightarrow$ Langages Init et Bad
Transition entre états $\longrightarrow$ Transducteur Post

But : Apprendre un langage L tel que :

- $\mathrm{Init} \subseteq L$
- $L \cap \mathrm{Bad} = \emptyset$
- $\mathrm{Post}(L) \subseteq L$

Habituellement : $L = \mathrm{Post}^*(\mathrm{Init})$ ou $L = \Sigma^* \setminus \mathrm{Pre}^*(\mathrm{Bad})$

Méthode : l'apprentissage actif

Algorithme de la boulangerie [3] :

Algorithme de la boulangerie [3] :

But : gestion de threads en zone critique, similaire à un système de ticket

Contexte

Exemple : algorithme de la boulangerie

Algorithme de la boulangerie [3] :

But : gestion de threads en zone critique, similaire à un système de ticket

3 états possibles par ticket :

Algorithme de la boulangerie [3] :

But : gestion de threads en zone critique, similaire à un système de ticket

3 états possibles par ticket :

- A : le ticket est disponible

Algorithme de la boulangerie [3] :

But : gestion de threads en zone critique, similaire à un système de ticket

3 états possibles par ticket :

- A : le ticket est disponible
- C : le ticket est en cours d'utilisation

Algorithme de la boulangerie [3] :

But : gestion de threads en zone critique, similaire à un système de ticket

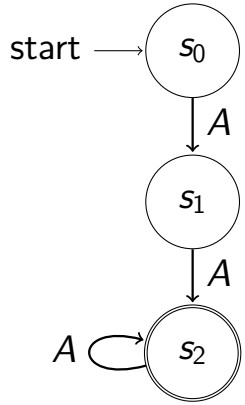
3 états possibles par ticket :

- A : le ticket est disponible
- C : le ticket est en cours d'utilisation
- W : le ticket est en cours d'utilisation et dans la file d'attente

Contexte

Exemple : algorithme de la boulangerie

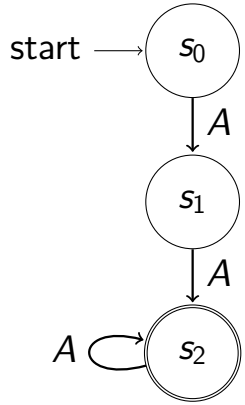
Init : AAA^* (au moins 2 A)



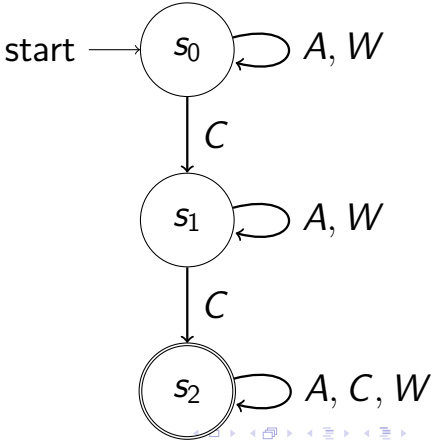
Contexte

Exemple : algorithme de la boulangerie

Init : AAA^* (au moins 2 A)



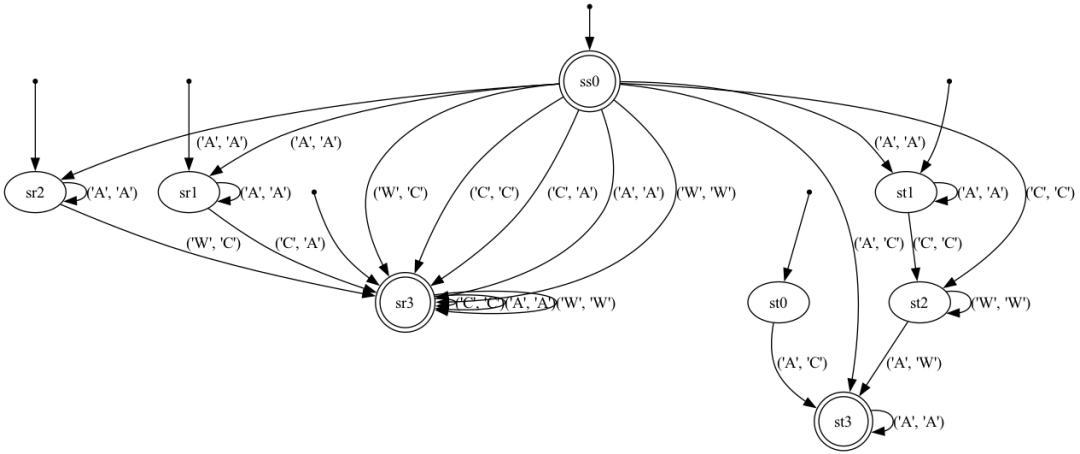
Bad : $((A + W)^*C)^2(A + W + C)^*$ (au moins 2 C)



Contexte

Exemple : algorithme de la boulangerie

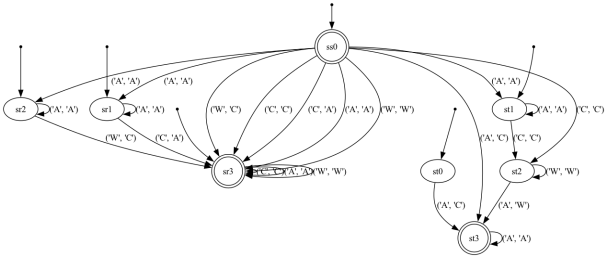
Transducteur :



Contexte

Exemple : algorithme de la boulangerie

Transducteur :

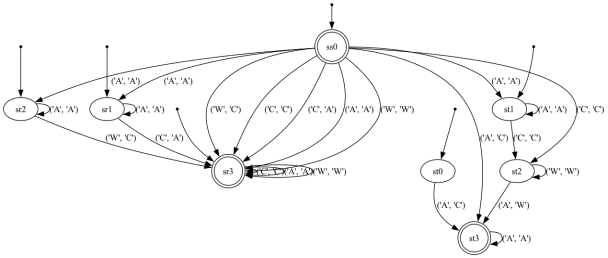


Transitions possibles :

Contexte

Exemple : algorithme de la boulangerie

Transducteur :



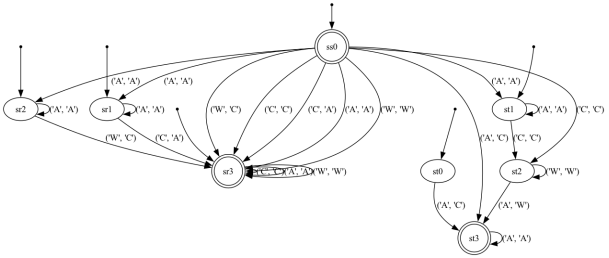
Transitions possibles :

- Aucun changement

Contexte

Exemple : algorithme de la boulangerie

Transducteur :



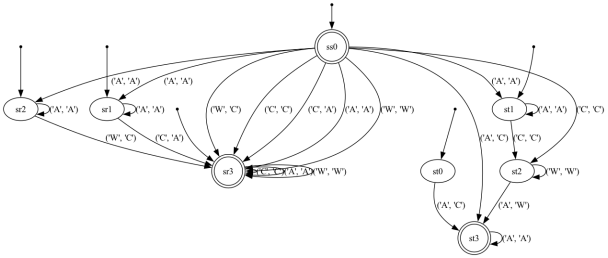
Transitions possibles :

- Aucun changement
- Ticket C devient A

Contexte

Exemple : algorithme de la boulangerie

Transducteur :



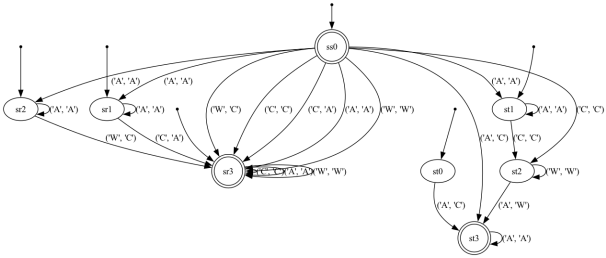
Transitions possibles :

- Aucun changement
- Ticket C devient A
- Premier ticket W devient C

Contexte

Exemple : algorithme de la boulangerie

Transducteur :



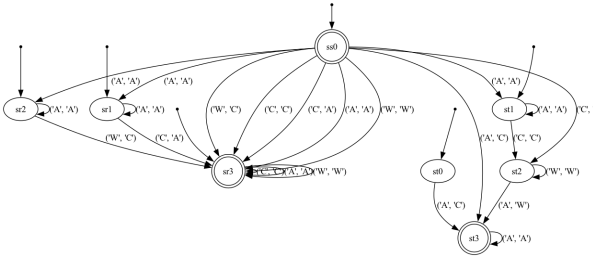
Transitions possibles :

- Aucun changement
- Ticket C devient A
- Premier ticket W devient C
- Ticket A après un C devient W

Contexte

Exemple : algorithme de la boulangerie

Transducteur :



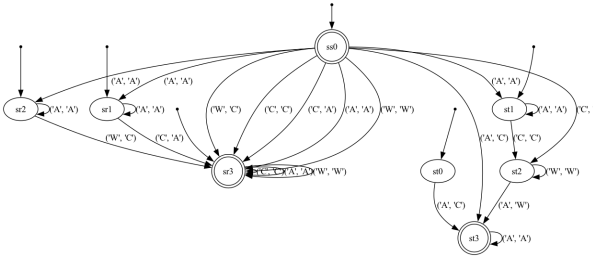
Transitions possibles :

- Aucun changement
- Ticket C devient A
- Premier ticket W devient C
- Ticket A après un C devient W
- Si aucun C ou W, premier ticket A devient C

Contexte

Exemple : algorithme de la boulangerie

Transducteur :



Transitions possibles :

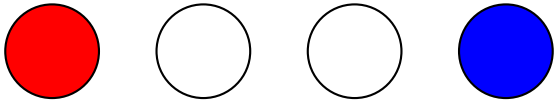
- Aucun changement
- Ticket C devient A
- Premier ticket W devient C
- Ticket A après un C devient W
- Si aucun C ou W, premier ticket A devient C

$$\mathrm{Post}^*(\mathrm{Init}) = (A^+ + A^*C)W^*A^*$$

Contexte

Problème : Equi-distance

Equi-distance :

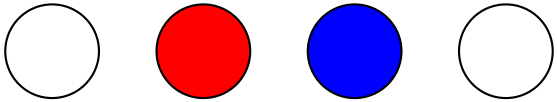


- 2 tokens : A et B séparés par des x

Contexte

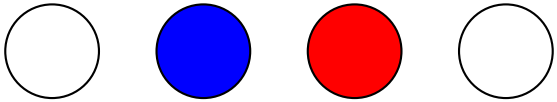
Problème : Equi-distance

Equi-distance :



- 2 tokens : A et B séparés par des x
- Peuvent se déplacer

Equi-distance :

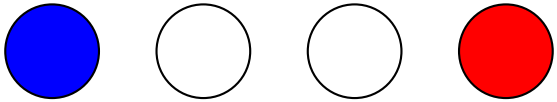


- 2 tokens : A et B séparés par des x
- Peuvent se déplacer
- Peuvent se croiser

Contexte

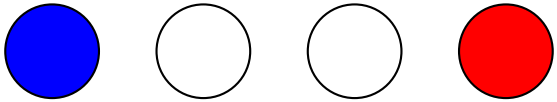
Problème : Equi-distance

Equi-distance :



- 2 tokens : A et B séparés par des x
- Peuvent se déplacer
- Peuvent se croiser
- Ne doivent pas atteindre le côté opposé

Equi-distance :



- 2 tokens : A et B séparés par des x
- Peuvent se déplacer
- Peuvent se croiser
- Ne doivent pas atteindre le côté opposé
- Bloqués si un seul x entre les tokens

Equi-distance :

- $\mathrm{Init} = Ax(xx)^*B$

Equi-distance :

- $\mathrm{Init} = Ax(xx)^*B$
- $\mathrm{Bad} = Bx^*A$

Equi-distance :

- $\mathrm{Init} = Ax(xx)^*B$
- $\mathrm{Bad} = Bx^*A$
- $\mathrm{Post}^*(\mathrm{Init}) = x^nAx(xx)^*Bx^n$

Equi-distance :

- $\mathrm{Init} = Ax(xx)^*B$
- $\mathrm{Bad} = Bx^*A$
- $\mathrm{Post}^*(\mathrm{Init}) = x^nAx(xx)^*Bx^n$
- $\mathrm{Pre}^*(\mathrm{Bad}) = x^nBx^*Ax^n + x^nA(xx)^*Bx^n$

Equi-distance :

- $\mathrm{Init} = Ax(xx)^*B$
- $\mathrm{Bad} = Bx^*A$
- $\mathrm{Post}^*(\mathrm{Init}) = x^nAx(xx)^*Bx^n$
- $\mathrm{Pre}^*(\mathrm{Bad}) = x^nBx^*Ax^n + x^nA(xx)^*Bx^n$

Langages non rationnels :

Contexte

Equi-distance :

- $\mathrm{Init} = Ax(xx)^*B$
- $\mathrm{Bad} = Bx^*A$
- $\mathrm{Post}^*(\mathrm{Init}) = x^nAx(xx)^*Bx^n$
- $\mathrm{Pre}^*(\mathrm{Bad}) = x^nBx^*Ax^n + x^nA(xx)^*Bx^n$

Langages non rationnels : apprentissage impossible !

Equi-distance :

- $\mathrm{Init} = Ax(xx)^*B$
- $\mathrm{Bad} = Bx^*A$
- $\mathrm{Post}^*(\mathrm{Init}) = x^nAx(xx)^*Bx^n$
- $\mathrm{Pre}^*(\mathrm{Bad}) = x^nBx^*Ax^n + x^nA(xx)^*Bx^n$

Langages non rationnels : apprentissage impossible !

Solution : ajout d'une réponse inconnue

Algorithme L_{ICE}

Sommaire

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'appartenance : mot w et liste de mots U

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'appartenance : mot w et liste de mots U

Théorie :

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'appartenance : mot w et liste de mots U

Théorie :

- $w \in \text{Post}^*(\text{Init})$: le mot w appartient au langage

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'appartenance : mot w et liste de mots U

Théorie :

- $w \in \mathrm{Post}^*(\mathrm{Init})$: le mot w appartient au langage
- $w \in \mathrm{Pre}^*(\mathrm{Bad})$: le mot w n'appartient pas au langage

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'appartenance : mot w et liste de mots U

Théorie :

- $w \in \text{Post}^*(\text{Init})$: le mot w appartient au langage
- $w \in \text{Pre}^*(\text{Bad})$: le mot w n'appartient pas au langage
- $\exists u \in U, u \in \text{Post}^*(w) : w \in L \Rightarrow u \in L$

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'appartenance : mot w et liste de mots U

Théorie :

- $w \in \text{Post}^*(\text{Init})$: le mot w appartient au langage
- $w \in \text{Pre}^*(\text{Bad})$: le mot w n'appartient pas au langage
- $\exists u \in U, u \in \text{Post}^*(w) : w \in L \Rightarrow u \in L$
- $\exists u \in U, u \in \text{Pre}^*(w) : u \in L \Rightarrow w \in L$

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'appartenance : mot w et liste de mots U

Théorie :

- $w \in \text{Post}^*(\text{Init})$: le mot w appartient au langage
- $w \in \text{Pre}^*(\text{Bad})$: le mot w n'appartient pas au langage
- $\exists u \in U, u \in \text{Post}^*(w) : w \in L \Rightarrow u \in L$
- $\exists u \in U, u \in \text{Pre}^*(w) : u \in L \Rightarrow w \in L$
- Sinon : appartenance inconnue

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'appartenance : mot w et liste de mots U

En pratique :

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'appartenance : mot w et liste de mots U

En pratique :

- $\text{Post}^*(\text{Init})$ et $\text{Pre}^*(\text{Bad})$ pas forcément calculables

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'appartenance : mot w et liste de mots U

En pratique :

- $\text{Post}^*(\text{Init})$ et $\text{Pre}^*(\text{Bad})$ pas forcément calculables
- $\text{Post}^*(w)$ et $\text{Pre}^*(w)$ le sont

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'appartenance : mot w et liste de mots U

En pratique :

- $\text{Post}^*(\text{Init})$ et $\text{Pre}^*(\text{Bad})$ pas forcément calculables
- $\text{Post}^*(w)$ et $\text{Pre}^*(w)$ le sont

- $\text{Pre}^*(w) \cap \text{Init} \neq \emptyset$: le mot w appartient au langage

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'appartenance : mot w et liste de mots U

En pratique :

- $\text{Post}^*(\text{Init})$ et $\text{Pre}^*(\text{Bad})$ pas forcément calculables
- $\text{Post}^*(w)$ et $\text{Pre}^*(w)$ le sont

- $\text{Pre}^*(w) \cap \text{Init} \neq \emptyset$: le mot w appartient au langage
- $\text{Post}^*(w) \cap \text{Bad} \neq \emptyset$: le mot w n'appartient pas au langage

Requêtes d'équivalence : automate $\mathcal{A}$

Requêtes d'équivalence : automate $\mathcal{A}$

- $\exists w,\ w \in \text{Init},\ w \notin \mathcal{L}(\mathcal{A})$: w est un contre-exemple

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'équivalence : automate $\mathcal{A}$

- $\exists w,\ w \in \text{Init},\ w \notin \mathcal{L}(\mathcal{A})$: w est un contre-exemple
- $\exists w,\ w \in \text{Bad},\ w \in \mathcal{L}(\mathcal{A})$: w est un contre-exemple

Algorithme L_{ICE}

Requêtes avec réponses inconnues

Requêtes d'équivalence : automate $\mathcal{A}$

- $\exists w,\ w \in \text{Init},\ w \notin \mathcal{L}(\mathcal{A})$: w est un contre-exemple
- $\exists w,\ w \in \text{Bad},\ w \in \mathcal{L}(\mathcal{A})$: w est un contre-exemple
- $\exists w,\ w \in \mathcal{L}(\mathcal{A}),\ \exists w' \in \text{Post}(w),\ w' \notin \text{Post}(\mathcal{L}(\mathcal{A}))$: relation $w \in L \Rightarrow w' \in L$ manquante

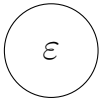
Premier cycle de requêtes d'appartenance pour l'algorithme de la boulangerie :

Algorithme L_{ICE}

Graphe d'appartenance inconnue

Premier cycle de requêtes d'appartenance pour l'algorithme de la boulangerie :

- $\varepsilon : \square$

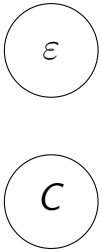


Algorithme L_{ICE}

Graphe d'appartenance inconnue

Premier cycle de requêtes d'appartenance pour l'algorithme de la boulangerie :

- $\varepsilon : \square$
- $C : \square$

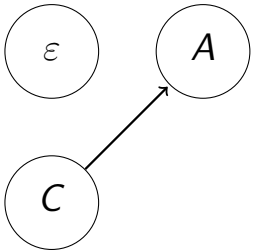


Algorithme L_{ICE}

Graphe d'appartenance inconnue

Premier cycle de requêtes d'appartenance pour l'algorithme de la boulangerie :

- $\varepsilon : \square$
- $C : \square$
- $A : C \in L \Rightarrow A \in L$

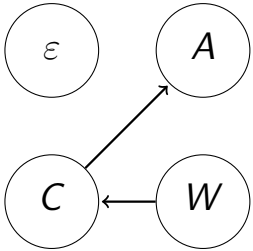


Algorithme L_{ICE}

Graphe d'appartenance inconnue

Premier cycle de requêtes d'appartenance pour l'algorithme de la boulangerie :

- $\varepsilon : \square$
- $C : \square$
- $A : C \in L \Rightarrow A \in L$
- $W : W \in L \Rightarrow C \in L$



Algorithme L_{ICE}

Table d'observation

Clauses SAT :

- x_w : w est accepté dans le langage

Algorithme L_{ICE}

Table d'observation

Clauses SAT :

- x_w : w est accepté dans le langage
- b_p : p est dans l'ensemble de préfixes

Algorithme L_{ICE}

Table d'observation

Clauses SAT :

- x_w : w est accepté dans le langage
- b_p : p est dans l'ensemble de préfixes
- $e_{p,a,p'}$: les lignes de $p \cdot a$ et p' sont équivalentes

Algorithme L_{ICE}

Table d'observation

Clauses SAT :

- x_w : w est accepté dans le langage
- b_p : p est dans l'ensemble de préfixes
- $e_{p,a,p'}$: les lignes de $p \cdot a$ et p' sont équivalentes

Conditions	Clauses SAT
	b_ε

Algorithme L_{ICE}

Table d'observation

Clauses SAT :

- x_w : w est accepté dans le langage
- b_p : p est dans l'ensemble de préfixes
- $e_{p,a,p'}$: les lignes de $p \cdot a$ et p' sont équivalentes

Conditions	Clauses SAT
	b_ε
$p \in P, a \in \Sigma, p \cdot a \in P$	$b_{p \cdot a} \rightarrow b_p$

Algorithme L_{ICE}

Table d'observation

Clauses SAT :

- x_w : w est accepté dans le langage
- b_p : p est dans l'ensemble de préfixes
- $e_{p,a,p'}$: les lignes de $p \cdot a$ et p' sont équivalentes

Conditions	Clauses SAT
	b_ε
$p \in P, a \in \Sigma, p \cdot a \in P$	$b_{p \cdot a} \rightarrow b_p$
$u_1 \in U, u_2 \in U_{u_1}$	$x_{u_1} \rightarrow x_{u_2}$

Algorithme L_{ICE}

Table d'observation

Clauses SAT :

- x_w : w est accepté dans le langage
- b_p : p est dans l'ensemble de préfixes
- $e_{p,a,p'}$: les lignes de $p \cdot a$ et p' sont équivalentes

Conditions	Clauses SAT
	b_ε
$p \in P, a \in \Sigma, p \cdot a \in P$	$b_{p \cdot a} \rightarrow b_p$
$u_1 \in U, u_2 \in U_{u_1}$	$x_{u_1} \rightarrow x_{u_2}$
$a \in \Sigma, s \in S, p, p' \in P^2, p' \neq p \cdot a$	$e_{p,a,p'} \rightarrow (x_{pas} \leftrightarrow x_{p's})$

Algorithme L_{ICE}

Table d'observation

Clauses SAT :

- x_w : w est accepté dans le langage
- b_p : p est dans l'ensemble de préfixes
- $e_{p,a,p'}$: les lignes de $p \cdot a$ et p' sont équivalentes

Conditions	Clauses SAT
	b_ε
$p \in P, a \in \Sigma, p \cdot a \in P$	$b_{p \cdot a} \rightarrow b_p$
$u_1 \in U, u_2 \in U_{u_1}$	$x_{u_1} \rightarrow x_{u_2}$
$a \in \Sigma, s \in S, p, p' \in P^2, p' \neq p \cdot a$	$e_{p,a,p'} \rightarrow (x_{pas} \leftrightarrow x_{p's})$
$p, p_1, p_2 \in P^3, a \in \Sigma, p_1 < p_2$	$e_{p,a,p_1} \rightarrow \neg e_{p,a,p_2}$

Algorithme L_{ICE}

Table d'observation

Clauses SAT :

- x_w : w est accepté dans le langage
- b_p : p est dans l'ensemble de préfixes
- $e_{p,a,p'}$: les lignes de $p \cdot a$ et p' sont équivalentes

Conditions	Clauses SAT
	b_ε
$p \in P, a \in \Sigma, p \cdot a \in P$	$b_{p \cdot a} \rightarrow b_p$
$u_1 \in U, u_2 \in U_{u_1}$	$x_{u_1} \rightarrow x_{u_2}$
$a \in \Sigma, s \in S, p, p' \in P^2, p' \neq p \cdot a$	$e_{p,a,p'} \rightarrow (x_{pas} \leftrightarrow x_{p's})$
$p, p_1, p_2 \in P^3, a \in \Sigma, p_1 < p_2$	$e_{p,a,p_1} \rightarrow \neg e_{p,a,p_2}$
$p \in P, a \in \Sigma$	$b_p \rightarrow \bigvee_{p' \in P} e_{p,a,p'}$

Algorithme L_{ICE}

Table d'observation

Clauses SAT :

- x_w : w est accepté dans le langage
- b_p : p est dans l'ensemble de préfixes
- $e_{p,a,p'}$: les lignes de $p \cdot a$ et p' sont équivalentes

Conditions	Clauses SAT
	b_ε
$p \in P, a \in \Sigma, p \cdot a \in P$	$b_{p \cdot a} \rightarrow b_p$
$u_1 \in U, u_2 \in U_{u_1}$	$x_{u_1} \rightarrow x_{u_2}$
$a \in \Sigma, s \in S, p, p' \in P^2, p' \neq p \cdot a$	$e_{p,a,p'} \rightarrow (x_{pas} \leftrightarrow x_{p's})$
$p, p_1, p_2 \in P^3, a \in \Sigma, p_1 < p_2$	$e_{p,a,p_1} \rightarrow \neg e_{p,a,p_2}$
$p \in P, a \in \Sigma$	$b_p \rightarrow \bigvee_{p' \in P} e_{p,a,p'}$
$p, p' \in P^2, a \in \Sigma$	$e_{p,a,p'} \rightarrow b_{p'}$

Algorithme L_{ICE}

Table d'observation

	ε
ε	$\square$
A	$\square_{\{C\}}$
C	$\square_{\{W\}}$
W	$\square$

Algorithme L_{ICE}

Table d'observation

	ε
ε	$\square$
A	$\square_{\{C\}}$
C	$\square_{\{W\}}$
W	$\square$

$\Longrightarrow$

	ε
ε	$\boxminus$
A	$\boxminus$
C	$\boxminus$
W	$\boxminus$

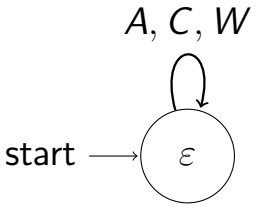
Algorithme L_{ICE}

Table d'observation

	ε
ε	$\square$
A	$\square_{\{C\}}$
C	$\square_{\{W\}}$
W	$\square$

$\Longrightarrow$

	ε
ε	⊟
A	⊟
C	⊟
W	⊟

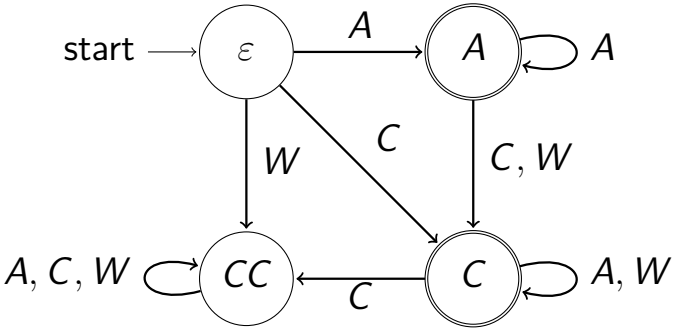


Résultats

Sommaire

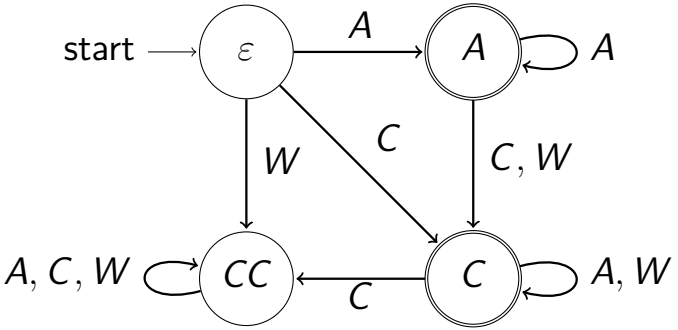
Résultats

Exemple : Algorithme de la boulangerie



Résultats

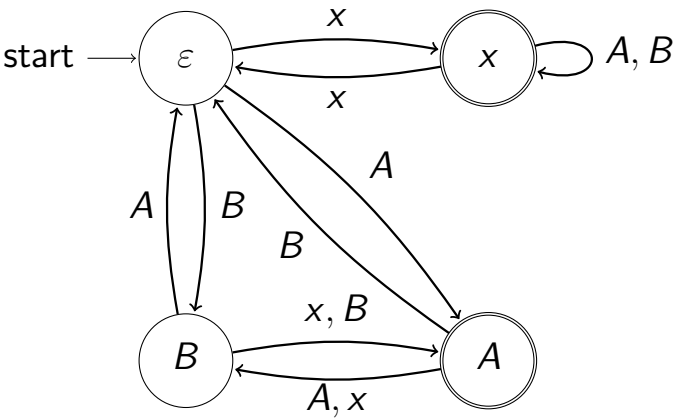
Exemple : Algorithme de la boulangerie



Langage trouvé : $(A^+ + A^*C)(A + W)^*$

Résultats

Exemple : Equi-dist



Résultats

Benchmarks

Modèle	Taille	Temps (s)	Appartenance	Équivalence
bakery	4	0.2	60	13
Burns	3	0.7	143	23
coffee can	3	≤ 0.2	51	7.1
coffee can v2	3	≤ 0.1	38	9.2
equi-dist	4	5.8	58	798
herman linear	2	≤ 0.1	5	1
herman ring	2	≤ 0.1	5	1
Israeli-Jalfon	2	≤ 0.1	11	2
Lehmann-Rabin	6.14	61.2	378.8	1269
LR philo	2	≤ 0.1	30	3.2
mux array	3	0.6	187	4
resource allocator	4	≤ 0.2	71	3

Bibliographie I

[1] Dana Angluin.
Learning regular sets from queries and counterexamples.
Information and Computation, 75(2) :87–106, November 1987.

[2] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili.
Regular Model Checking.
In E. Allen Emerson and Aravinda Prasad Sistla, editors, *Computer Aided Verification*, pages 403–418, Berlin, Heidelberg, 2000. Springer.

Bibliographie II

[3] Yu-Fang Chen, Chih-Duo Hong, Anthony W. Lin, and Philipp Ruemmer.
Learning to Prove Safety over Parameterised Concurrent Systems (Full Version), October 2017.
arXiv :1709.07139 [cs].

[4] Mark Moeller, Thomas Wiener, Alaia Solko-Breslin, Caleb Koch, Nate Foster, and Alexandra Silva.
Automata Learning with an Incomplete Teacher.
2023.