Active Learning Techniques for Pomset Recognisers

Présentation de fin de semestre

Edgar Delaporte, encadré par Amazigh Amrane et Adrien Pommellet

14 Janvier 2025

Contexte

- Les automates finis permettent de modéliser les systèmes ordonnées (flot de contrôle d'un programme, protocole de communication, séquence de génome, ...).

Contexte

- Les automates finis permettent de modéliser les systèmes ordonnées (flot de contrôle d'un programme, protocole de communication, séquence de génome, ...).

- Ils sont peu performants lorsqu'il s'agit de décrire des systèmes concurrents (programme parallèle, système non-déterministe).

Contexte

- Les automates finis permettent de modéliser les systèmes ordonnées (flot de contrôle d'un programme, protocole de communication, séquence de génome, ...).
- Ils sont peu performants lorsqu'il s'agit de décrire des systèmes concurrents (programme parallèle, système non-déterministe).
- On modélise alors de tels systèmes par des **Pomsets** et l'on étudie les machines qui les reconnaissent.

Contexte

- Les automates finis permettent de modéliser les systèmes ordonnées (flot de contrôle d'un programme, protocole de communication, séquence de génome, ...).
- Ils sont peu performants lorsqu'il s'agit de décrire des systèmes concurrents (programme parallèle, système non-déterministe).
- On modélise alors de tels systèmes par des **Pomsets** et l'on étudie les machines qui les reconnaissent.
- Étendre l'état de l'art sur l'**Apprentissage Actif** vers les machines reconnaissant les pomsets permettrait de pouvoir extraire un modèle d'un système parallèle -> ouvre la voie vers vérification, model checking, ...

État de l'art

- Des machines reconnaissant les pomsets ont été décrites [1] [2].

1. Bedon, « Branching Automata and Pomset Automata »
2. Fahrenberg et al., « A Kleene Theorem for Higher-Dimensional Automata »
3. Lodaya et Weil, « A Kleene Iteration for Parallelism »
4. Heerdt et al., « Learning Pomset Automata »

État de l'art

- Des machines reconnaissant les pomsets ont été décrites [1] [2].
- Des propriétés intéressantes sur ces machines et leurs langages ont été démontrées (th. de Myhill-Nerode [3]).

1. Bedon, « Branching Automata and Pomset Automata »
2. Fahrenberg et al., « A Kleene Theorem for Higher-Dimensional Automata »
3. Lodaya et Weil, « A Kleene Iteration for Parallelism »
4. Heerdt et al., « Learning Pomset Automata »

État de l'art

- Des machines reconnaissant les pomsets ont été décrites [1] [2].
- Des propriétés intéressantes sur ces machines et leurs langages ont été démontrées (th. de Myhill-Nerode [3]).
- Des algorithmes d'apprentissage actif de telles machines ont été proposés [4].

1. Bedon, « Branching Automata and Pomset Automata »
2. Fahrenberg et al., « A Kleene Theorem for Higher-Dimensional Automata »
3. Lodaya et Weil, « A Kleene Iteration for Parallelism »
4. Heerdt et al., « Learning Pomset Automata »

Objectifs

- Adapter L^λ [5] aux machines de pomset.

5. Howar et Steffen, « Active Automata Learning as Black-Box Search and Lazy Partition Refinement »

Objectifs

- Adapter L^λ [5] aux machines de pomset.
- Proposer des méthodes de comparaisons des machines utilisées.

5. Howar et Steffen, « Active Automata Learning as Black-Box Search and Lazy Partition Refinement »

Objectifs

- Adapter L^λ [5] aux machines de pomset.
- Proposer des méthodes de comparaisons des machines utilisées.
- Améliorer le traitement des contre-exemples.

5. Howar et Steffen, « Active Automata Learning as Black-Box Search and Lazy Partition Refinement »

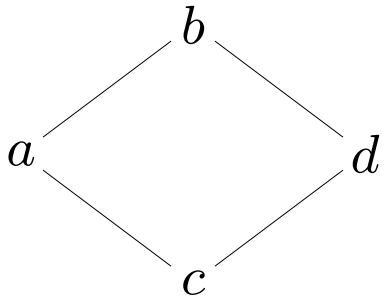
Objectifs

- Adapter L^λ [5] aux machines de pomset.
- Proposer des méthodes de comparaisons des machines utilisées.
- Améliorer le traitement des contre-exemples.
- Générer des cas de tests.

5. Howar et Steffen, « Active Automata Learning as Black-Box Search and Lazy Partition Refinement »

Pomsets

- **Poset** : $(A, <, i)$.

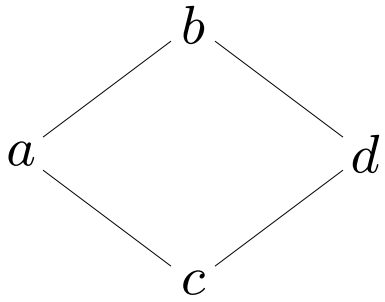


Poset p = a(b||c)d

- **Ordre partiel** : relation d'ordre non fortement connectée ($a \geqslant b \bigoplus b > a$).

Pomsets

- **Poset** : $(A, <, i)$.



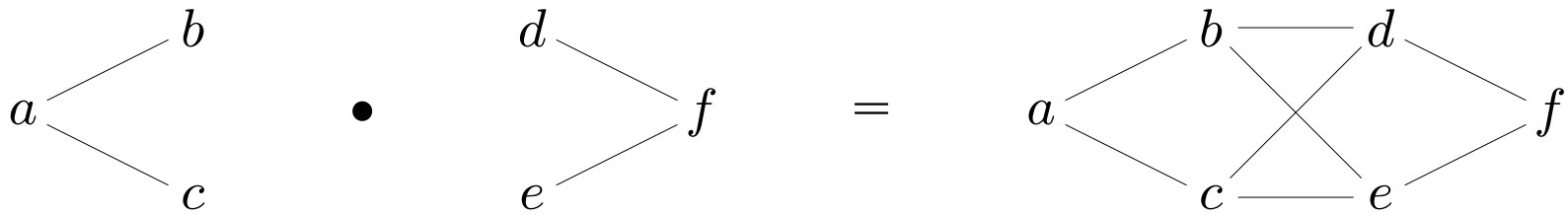
Poset p = a(b||c)d

- **Ordre partiel** : relation d'ordre non fortement connectée ($a \geqslant b \bigoplus b > a$).

- **Pomset** : classe d'équivalence de posets à isomorphisme près.

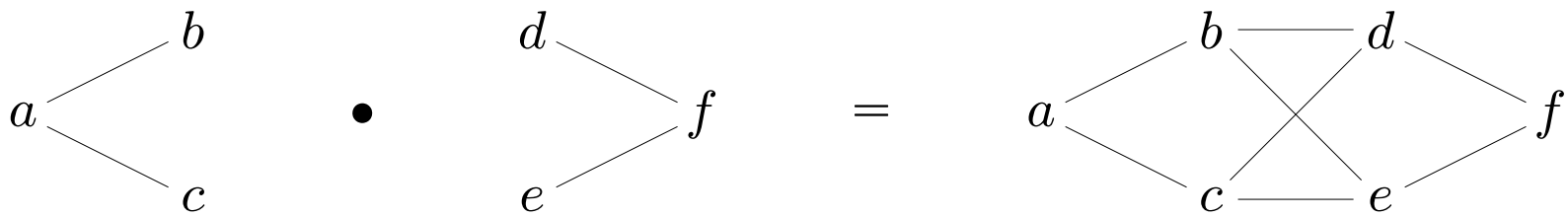
Pomsets - opérations

- produit séquentiel (concaténation) :

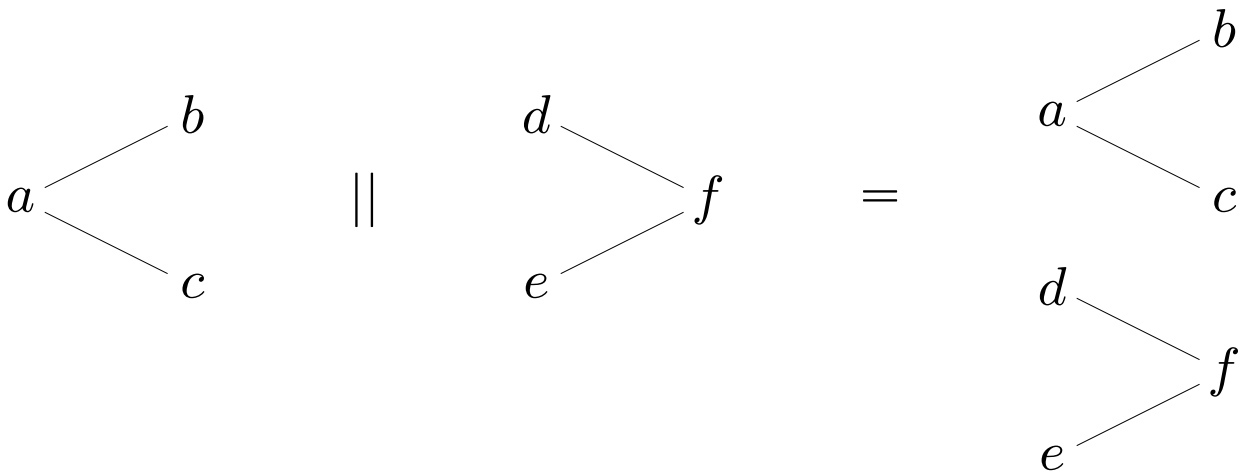


Pomsets - opérations

- produit séquentiel (concaténation) :

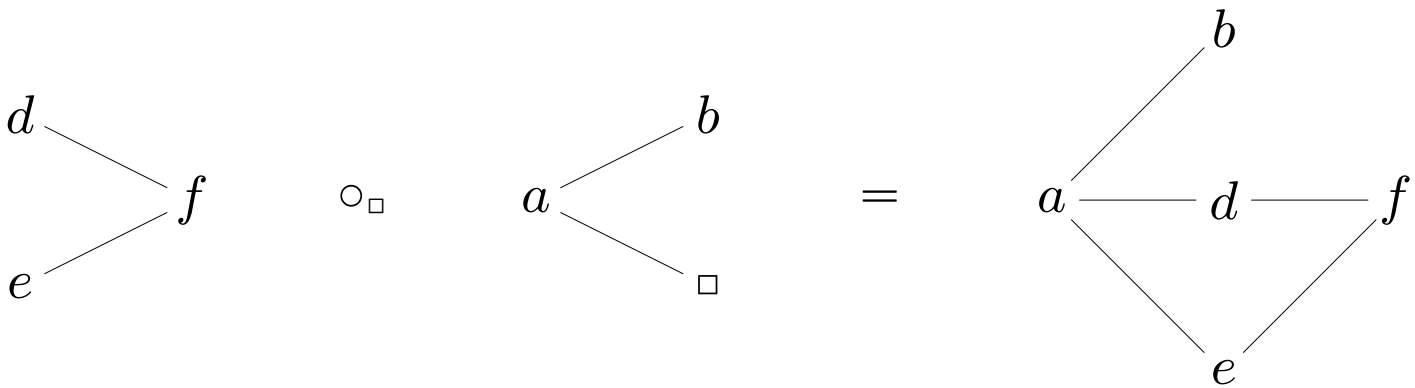


- produit parallèle :

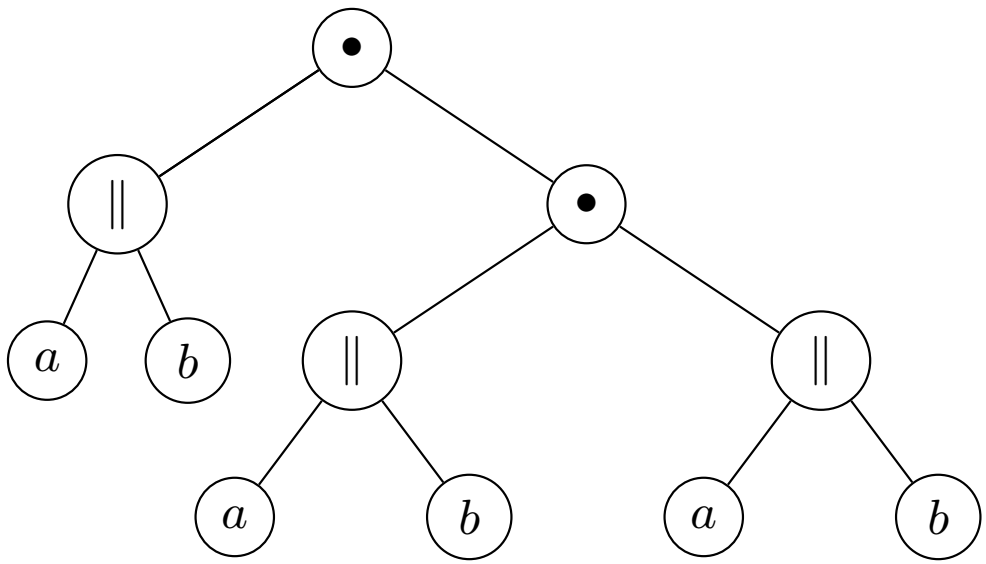


Pomsets - substitution

- $p2 \circ_{\square} p1$.
- Remplacer toutes les occurrences de $\square$ dans $p1$ par $p2$.



Pomsets - arbre syntaxique



$$(a||b) \bullet (a||b) \bullet (a||b)$$

Pomset Recognisers

- **Bimonoide** : $(M, \odot, \oplus, e)$.

Pomset Recognisers

- **Bimonoide** : $(M, \odot, \oplus, e)$.
- **Pomset recogniser** : $(M, \odot, \oplus, e, F, i)$.

Pomset Recognisers

- **Bimonoide** : $(M, \odot, \oplus, e)$.
- **Pomset recogniser** : $(M, \odot, \oplus, e, F, i)$.
- Les **PR** sont des automates d'arbre finis de degré 2.

Pomset Recognisers

- **Bimonoide** : $(M, \odot, \oplus, e)$.
- **Pomset recogniser** : $(M, \odot, \oplus, e, F, i)$.
- Les **PR** sont des automates d'arbre finis de degré 2.
- Requête d'appartenance d'un pomset p par évaluation de son arbre syntaxique.

Pomset Recognisers - exemple

$\odot$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$
q_a	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_a
q_b	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_b
q_1	$q_\perp$	$q_\perp$	q_1	$q_\perp$	q_1
$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$
$\mathbf{1}$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$

$\oplus$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$
q_a	$q_\perp$	q_1	$q_\perp$	$q_\perp$	q_a
q_b	q_1	$q_\perp$	$q_\perp$	$q_\perp$	q_b
q_1	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_1
$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$
$\mathbf{1}$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$

$F = \{q_1, \mathbf{1}\}$

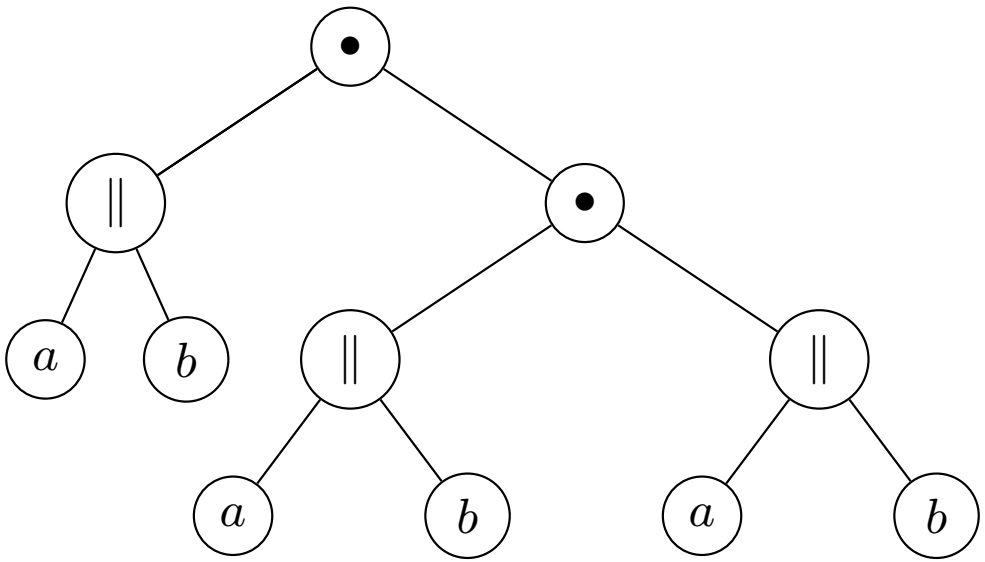
$i(a) = q_a,\ i(b) = q_b$

Reconnaît $L = (a||b)^* = \{\epsilon, a||b, (a||b)(a||b), (a||b)(a||b)(a||b), ...\}$.

Requête d'appartenance

$\odot$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$
q_a	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_a
q_b	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_b
q_1	$q_\perp$	$q_\perp$	q_1	$q_\perp$	q_1
$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$
$\mathbf{1}$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$

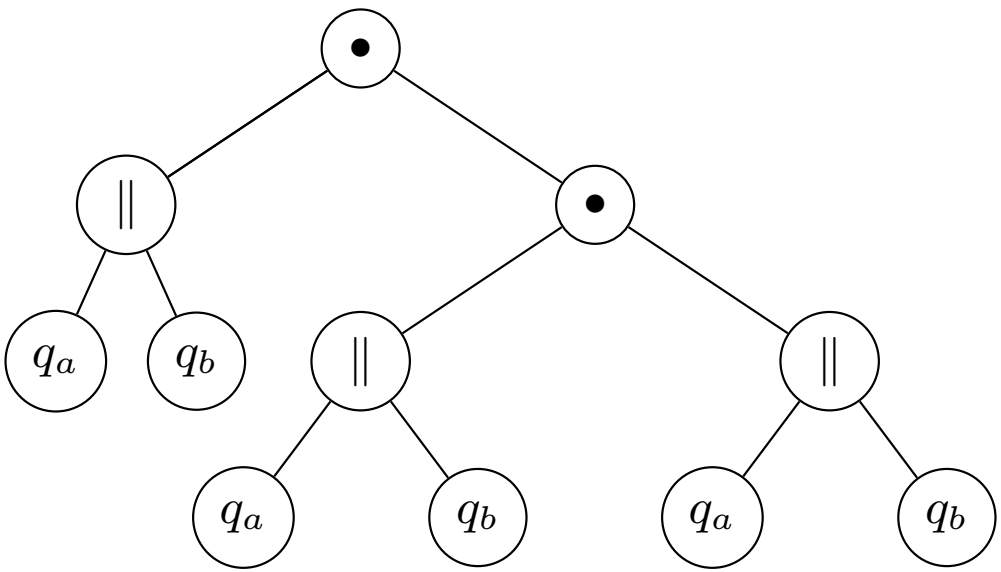
$\oslash$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$
q_a	$q_\perp$	q_1	$q_\perp$	$q_\perp$	q_a
q_b	q_1	$q_\perp$	$q_\perp$	$q_\perp$	q_b
q_1	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_1
$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$
$\mathbf{1}$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$



Requête d'appartenance

$\odot$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$
q_a	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_a
q_b	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_b
q_1	$q_\perp$	$q_\perp$	q_1	$q_\perp$	q_1
$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$
$\mathbf{1}$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$

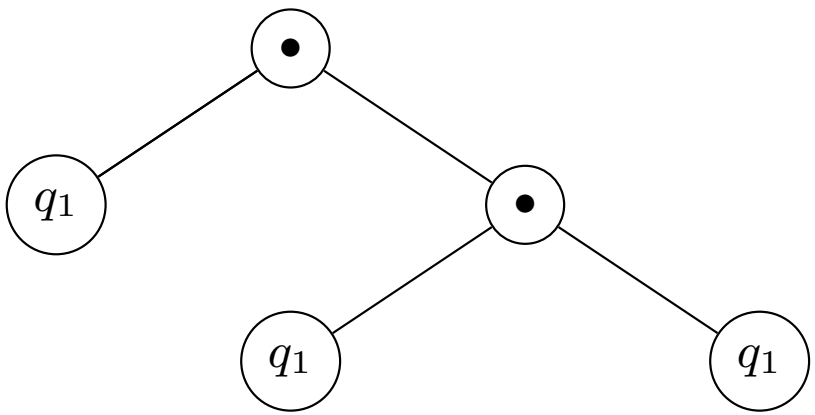
$\oslash$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$
q_a	$q_\perp$	q_1	$q_\perp$	$q_\perp$	q_a
q_b	q_1	$q_\perp$	$q_\perp$	$q_\perp$	q_b
q_1	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_1
$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$
$\mathbf{1}$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$



Requête d'appartenance

$\odot$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$
q_a	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_a
q_b	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_b
q_1	$q_\perp$	$q_\perp$	q_1	$q_\perp$	q_1
$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$
$\mathbf{1}$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$

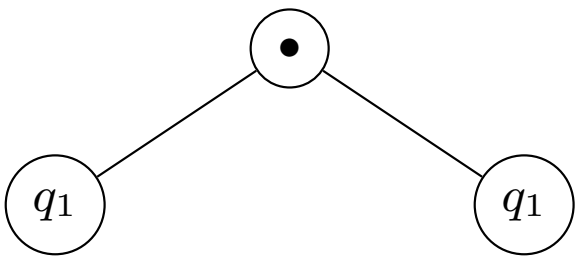
$\oslash$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$
q_a	$q_\perp$	q_1	$q_\perp$	$q_\perp$	q_a
q_b	q_1	$q_\perp$	$q_\perp$	$q_\perp$	q_b
q_1	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_1
$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$
$\mathbf{1}$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$



Requête d'appartenance

$\odot$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$
q_a	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_a
q_b	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_b
q_1	$q_\perp$	$q_\perp$	q_1	$q_\perp$	q_1
$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$
$\mathbf{1}$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$

$\oslash$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$
q_a	$q_\perp$	q_1	$q_\perp$	$q_\perp$	q_a
q_b	q_1	$q_\perp$	$q_\perp$	$q_\perp$	q_b
q_1	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_1
$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$
$\mathbf{1}$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$



Requête d'appartenance

$\odot$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$
q_a	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_a
q_b	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_b
q_1	$q_\perp$	$q_\perp$	q_1	$q_\perp$	q_1
$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$
$\mathbf{1}$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$

$\oplus$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$
q_a	$q_\perp$	q_1	$q_\perp$	$q_\perp$	q_a
q_b	q_1	$q_\perp$	$q_\perp$	$q_\perp$	q_b
q_1	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	q_1
$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$	$q_\perp$
$\mathbf{1}$	q_a	q_b	q_1	$q_\perp$	$\mathbf{1}$

Relation de Myhill-Nerode

- Un pomset c contenant une seule occurrence de $\square$ est un **contexte distingueur** des pomsets p_1 et p_2 de L ssi :

$$p_1 \ o_\square \ c \in L \neq p_2 \ o_\square \ c \in L$$

Relation de Myhill-Nerode

- Un pomset c contenant une seule occurrence de $\square$ est un **contexte distingueur** des pomsets p_1 et p_2 de L ssi :

$$p_1 \ o_\square \ c \in L \neq p_2 \ o_\square \ c \in L$$

- **Relation de Myhill-Nerode** :

$$\forall (p_1, p_2) \in L^2, p_1 \sim_L p_2 \iff \nexists c \in C, p_1 \ o_\square \ c \in L \neq p_2 \ o_\square \ c \in L$$

Relation de Myhill-Nerode

- Un pomset c contenant une seule occurrence de $\square$ est un **contexte distingueur** des pomsets p_1 et p_2 de L ssi :

$$p_1 \ o_\square \ c \in L \neq p_2 \ o_\square \ c \in L$$

- **Relation de Myhill-Nerode** :

$$\forall (p_1, p_2) \in L^2, p_1 \sim_L p_2 \iff \nexists c \in C, p_1 \ o_\square \ c \in L \neq p_2 \ o_\square \ c \in L$$

- L reconnaissable par un **PR** ssi $\sim_L$ est d'**ordre fini**.

Relation de Myhill-Nerode

- Un pomset c contenant une seule occurrence de $\square$ est un **contexte distingueur** des pomsets p_1 et p_2 de L ssi :

$$p_1 \ o_\square \ c \in L \neq p_2 \ o_\square \ c \in L$$

- **Relation de Myhill-Nerode** :

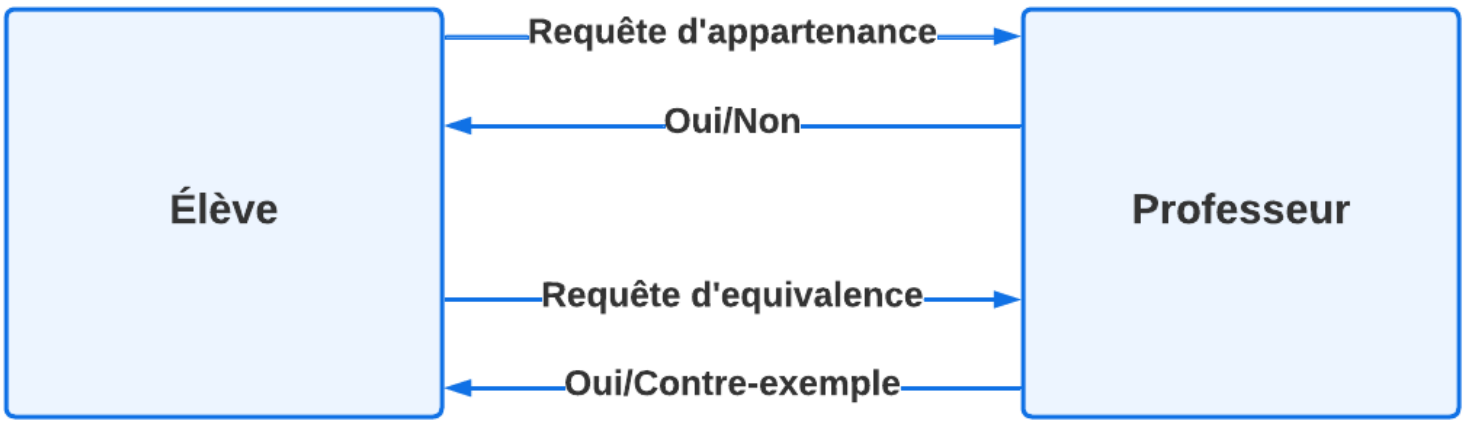
$$\forall (p_1, p_2) \in L^2, p_1 \sim_L p_2 \iff \nexists c \in C, p_1 \ o_\square \ c \in L \neq p_2 \ o_\square \ c \in L$$

- L reconnaissable par un **PR** ssi $\sim_L$ est d'**ordre fini**.

- Condition nécessaire et suffisante pour appliquer des méthodes d'apprentissage actif sur les **PR**.

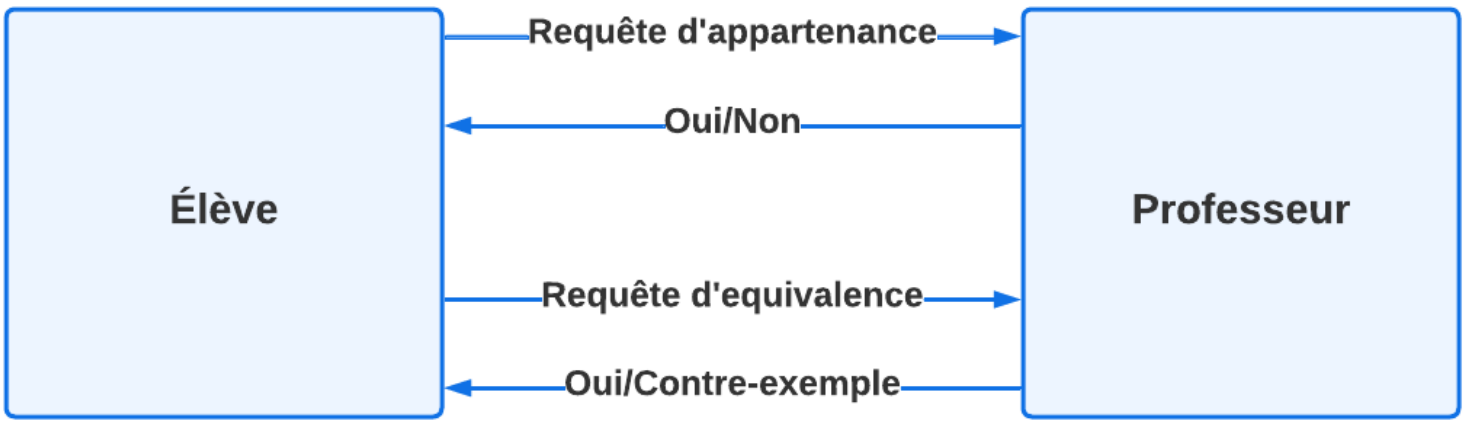
Apprentissage actif

- Apprendre une machine à états minimale d'un **MAT** :



Apprentissage actif

- Apprendre une machine à états minimale d'un **MAT** :



- **Objectif** : Trouver les classes d'équivalence de Myhill-Nerode en le moins de requêtes possible.

L^λ

- L^* a déjà été étendu aux **PR**.

L^λ

- L^* a déjà été étendu aux **PR**.
- Beaucoup de calculs inutiles.

L^λ

- L^* a déjà été étendu aux **PR**.
- Beaucoup de calculs inutiles.
- Dans le cas des mots, L^λ permet d'apprendre le MAT avec moins de requêtes d'appartenance.

L^λ

- L^* a déjà été étendu aux **PR**.
- Beaucoup de calculs inutiles.
- Dans le cas des mots, L^λ permet d'apprendre le MAT avec moins de requêtes d'appartenance.
- On se propose d'étendre cet algorithme aux **PR**.

L^λ - structures de données

- S : ensemble des **séquences d'accès**.

L^λ - structures de données

- S : ensemble des **séquences d'accès**.
- S^+ : Composition des éléments de S avec les lettres de l'alphabet Σ.

L^λ - structures de données

- S : ensemble des **séquences d'accès**.
- S^+ : Composition des éléments de S avec les lettres de l'alphabet Σ.
- $\mathcal{B} = \{B_1, B_2, B_3, ...\}$: partition de $S \cup S^+$, ensembles de sous-pomsets équivalents.

L^λ - structures de données

- S : ensemble des **séquences d'accès**.
- S^+ : Composition des éléments de S avec les lettres de l'alphabet Σ.
- $\mathcal{B} = \{B_1, B_2, B_3, ...\}$: partition de $S \cup S^+$, ensembles de sous-pomsets équivalents.
- $\mathcal{D}$: **arbre de discrimination**.

L^λ - opérations

- consistance, associativité, acuité → **PR**.

L^λ - opérations

- consistance, associativité, acuité $\rightarrow$ **PR**.
- **Expand**(p) ajoute un $p \in S^+$ a S.

L^λ - opérations

- consistance, associativité, acuité → **PR**.
- **Expand**(p) ajoute un $p \in S^+$ a S.
- **Refine**(B, c) "casse" un B en B_1 et B_2.

L^λ - opérations

- consistance, associativité, acuité → **PR**.
- **Expand**(p) ajoute un $p \in S^+$ a S.
- **Refine**(B, c) "casse" un B en B_1 et B_2.
- **MakeConsistent**() et **MakeAssoc**() trouvent les raffinements utiles.

Équivalence

- Nous avons besoin de pouvoir comparer notre hypothèse au modèle (cas d'arrêt). Deux cas de figure :
  - Le modèle est un **PR** "visible".
  - Le modèle est inconnu et/ou pas un **PR**.

Équivalence - cas idéal

- Si le modèle T est un **PR** connu, on peut utiliser :

$$L(T) = L(H) \iff \left(L(T) \cap \overline{L(H)} = \varnothing\right) \wedge \left(L(H) \cap \overline{L(T)} = \varnothing\right)$$

Équivalence - cas idéal

- Si le modèle T est un **PR** connu, on peut utiliser :

$$L(T) = L(H) \iff \left(L(T) \cap \overline{L(H)} = \varnothing\right) \wedge \left(L(H) \cap \overline{L(T)} = \varnothing\right)$$

- Intersection, complémentation et vacuité décidables en temps polynomial.

Équivalence - cas idéal

- Si le modèle T est un **PR** connu, on peut utiliser :

$$L(T) = L(H) \iff \left(L(T) \cap \overline{L(H)} = \varnothing\right) \wedge \left(L(H) \cap \overline{L(T)} = \varnothing\right)$$

- Intersection, complémentation et vacuité décidables en temps polynomial.

- Contre-exemple trivial à extraire dans le cas négatif.

Équivalence - cas idéal

- Si le modèle T est un **PR** connu, on peut utiliser :

$$L(T) = L(H) \iff \left(L(T) \cap \overline{L(H)} = \varnothing\right) \wedge \left(L(H) \cap \overline{L(T)} = \varnothing\right)$$

- Intersection, complémentation et vacuité décidables en temps polynomial.
- Contre-exemple trivial à extraire dans le cas négatif.
- Algorithme "non réaliste" puisque nécessite de connaître le résultat à l'avance.

Équivalence - méthode générale

- **W-method** : simuler les requêtes d'équivalence avec des requêtes d'appartenance.

Équivalence - méthode générale

- **W-method** : simuler les requêtes d'équivalence avec des requêtes d'appartenance.

- **State cover** : ensemble contenant ϵ et une **séquence d'accès** pour chaque état d'un **PR**.

Équivalence - méthode générale

- **W-method** : simuler les requêtes d'équivalence avec des requêtes d'appartenance.
- **State cover** : ensemble contenant ϵ et une **séquence d'accès** pour chaque état d'un **PR**.
- **Characterisation set** : ensemble de contextes qui suffisent à distinguer les états d'un **PR**.

Équivalence - méthode générale

- **W-method** : simuler les requêtes d'équivalence avec des requêtes d'appartenance.
- **State cover** : ensemble contenant ϵ et une **séquence d'accès** pour chaque état d'un **PR**.
- **Characterisation set** : ensemble de contextes qui suffisent à distinguer les états d'un **PR**.
- On extrapole un **state cover** de T à partir de celui de H : L^P_{k+1}.

Équivalence - méthode générale

- **W-method** : simuler les requêtes d'équivalence avec des requêtes d'appartenance.
- **State cover** : ensemble contenant ϵ et une **séquence d'accès** pour chaque état d'un **PR**.
- **Characterisation set** : ensemble de contextes qui suffisent à distinguer les états d'un **PR**.
- On extrapole un **state cover** de T à partir de celui de H : L^P_{k+1}.
- $L(T) = L(H)$ ssi T et H sont d'accords sur tous les pomsets de L^P_{k+1}.

Équivalence - méthode générale

- **W-method** : simuler les requêtes d'équivalence avec des requêtes d'appartenance.
- **State cover** : ensemble contenant ϵ et une **séquence d'accès** pour chaque état d'un **PR**.
- **Characterisation set** : ensemble de contextes qui suffisent à distinguer les états d'un **PR**.
- On extrapole un **state cover** de T à partir de celui de H : L^P_{k+1}.
- $L(T) = L(H)$ ssi T et H sont d'accords sur tous les pomsets de L^P_{k+1}.
- Nécessite d'estimer correctement k.

Traitement des contre-exemples

- On veut pouvoir extraire un maximum d'information d'un contre-exemple w.

Traitement des contre-exemples

- On veut pouvoir extraire un maximum d'information d'un contre-exemple w.

- **Rivest-Schapire** : faire évaluer partiellement w par H en remplaçant des sous-pomsets par leur **séquence d'accès** jusqu'à trouver un désaccord entre T et H (**point de rupture**).

Traitement des contre-exemples

- On veut pouvoir extraire un maximum d'information d'un contre-exemple w.

- **Rivest-Schapire** : faire évaluer partiellement w par H en remplaçant des sous-pomsets par leur **séquence d'accès** jusqu'à trouver un désaccord entre T et H (**point de rupture**).

- Si les deux fils d'un **point de rupture** provoquent un conflit, il est dit **effectif** et permet de trouver un nouveau raffinement.

Traitement des contre-exemples

- On veut pouvoir extraire un maximum d'information d'un contre-exemple w.
- **Rivest-Schapire** : faire évaluer partiellement w par H en remplaçant des sous-pomsets par leur **séquence d'accès** jusqu'à trouver un désaccord entre T et H (**point de rupture**).
- Si les deux fils d'un **point de rupture** provoquent un conflit, il est dit **effectif** et permet de trouver un nouveau raffinement.
- On trouve toujours un **point de rupture effectif** en explorant une seule branche de l'arbre syntaxique de w.

L^λ - exemple : $L = (a||b)^*$

- $S = \{\}$
- $S^+ = \{\}$
- $\mathcal{B} = \{$
  $\}$

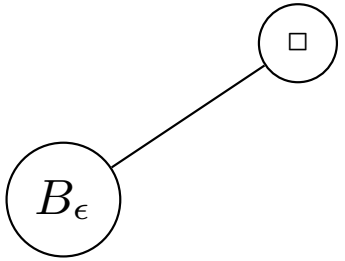
$\mathcal{D}$:

L^λ - exemple : Expand(ϵ)

- $S = \{\epsilon\}$
- $S^+ = \{a, b\}$
- $\mathcal{B} = \{$
  - $B_\epsilon = \{\epsilon\}$

  $\}$

$\mathcal{D}$:

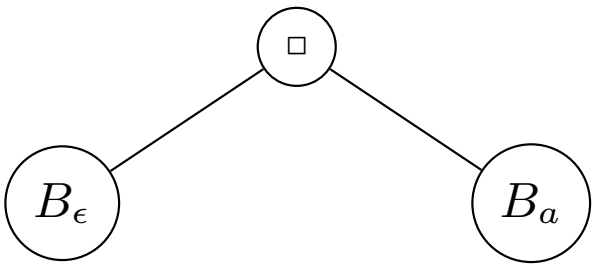


L^λ - exemple : Expand(a)

- $S = \{\epsilon, a\}$
- $S^+ = \{b, ab, a||b, ...\}$
- $\mathcal{B} = \{$
  - $B_\epsilon = \{\epsilon, a||b\}$
  - $B_a = \{a, b, ab, ...\}$

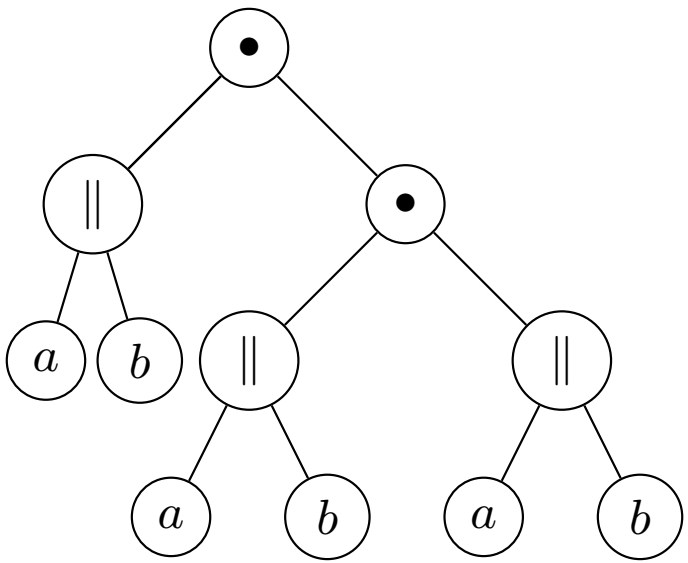
  $\}$

$\mathcal{D}$:



L^λ - exemple : traitement du contre-exemple

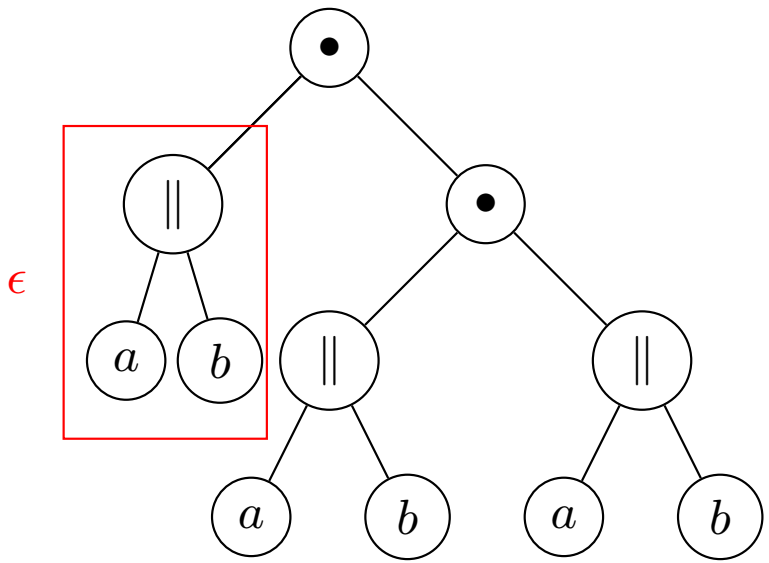
contre-exemple : $(a||b)(a||b)(a||b)$



- $c = \square$
- $z = (a||b)(a||b)(a||b)$

L^λ - exemple : traitement du contre-exemple

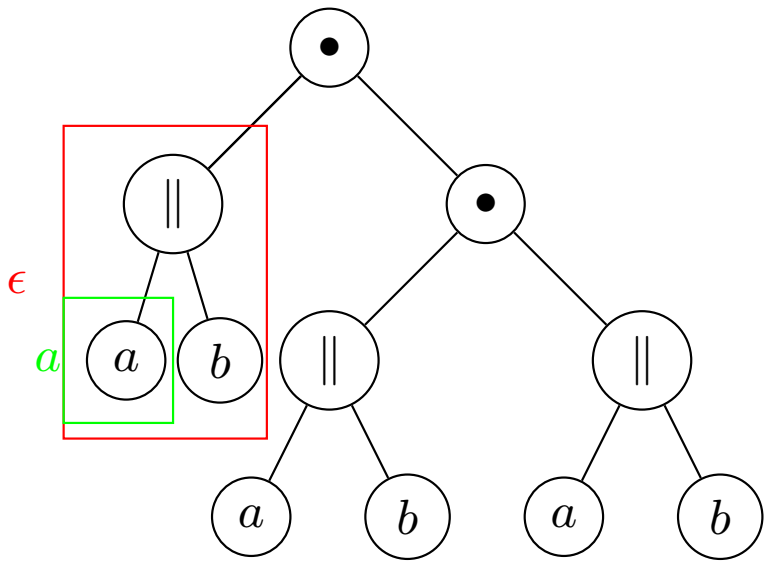
contre-exemple : $(a||b)(a||b)(a||b)$



- $c = \square(a||b)(a||b)$
- $z = (a||b)$

L^λ - exemple : traitement du contre-exemple

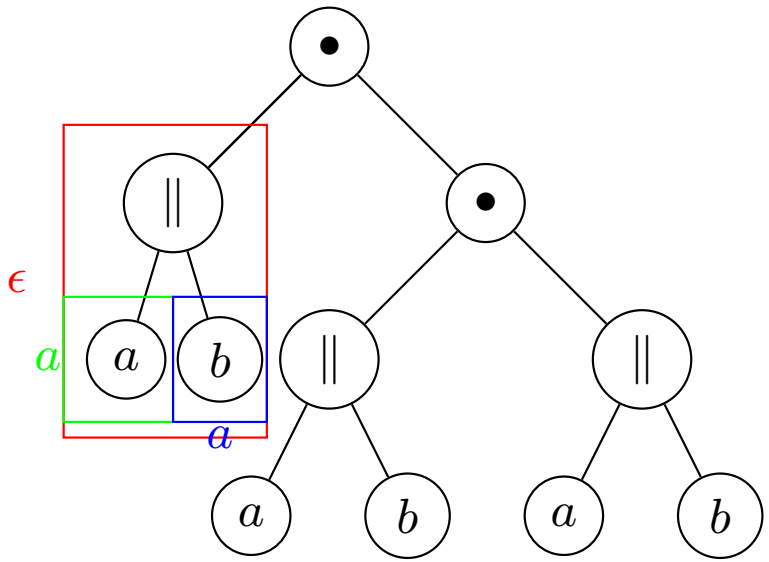
contre-exemple : $(a||b)(a||b)(a||b)$



- $c = (a||\square)(a||b)(a||b)$
- $z = b$

L^λ - exemple : traitement du contre-exemple

contre-exemple : $(a||b)(a||b)(a||b)$



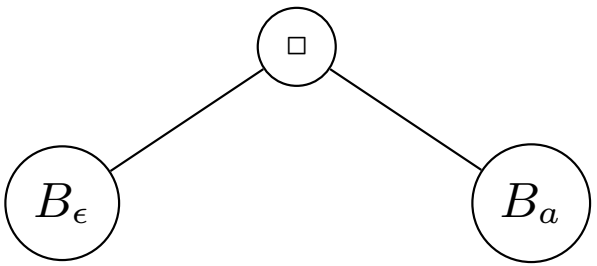
- $c = (a||\square)(a||b)(a||b)$
- $z = b$

L^λ - exemple : Expand(b)

- $S = \{\epsilon, a, b\}$
- $S^+ = \{a||b, ab, aa, ...\}$
- $\mathcal{B} = \{$
  - $B_\epsilon = \{\epsilon, a||b\}$
  - $B_a = \{a, b, ab, ...\}$

  $\}$

$\mathcal{D}$:

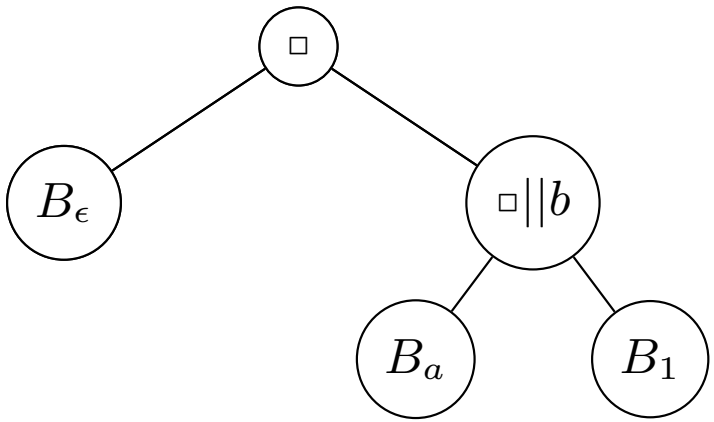


L^λ - exemple : Problème de consistance : Refine(B_a,$\square||b$)

- $S = \{\epsilon, a, b\}$
- $S^+ = \{a||b, ab, aa, ...\}$
- $\mathcal{B} = \{$
  - $B_\epsilon = \{\epsilon, a||b\}$
  - $B_a = \{a\}$
  - $B_1 = \{b, ab, aa, a||a, ...\}$

  $\}$

$\mathcal{D}$:

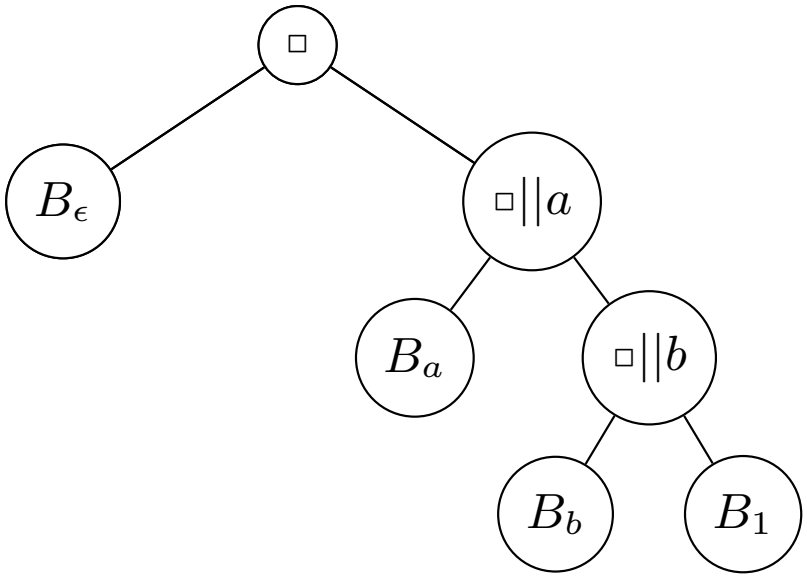


L^λ - exemple : Problème d'associativité : Refine(B_1,$a||\square$)

- $S = \{\epsilon, a, b\}$
- $S^+ = \{a||b, ab, aa, ...\}$
- $\mathcal{B} = \{$
  - $B_\epsilon = \{\epsilon, a||b\}$
  - $B_a = \{a\}$
  - $B_b = \{b\}$
  - $B_1 = \{ab, aa, a||a, ...\}$

  $\}$

$\mathcal{D}$:

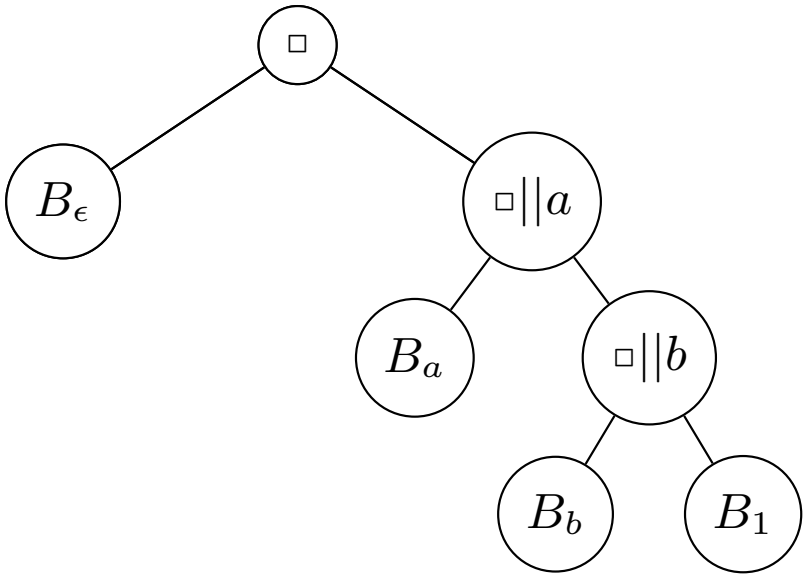


L^λ - exemple : Expand$(a||a)$

- $S = \{\epsilon, a, b, a||a\}$
- $S^+ = \{a||b, ab, aa, a(a||a), ...\}$
- $\mathcal{B} = \{$
  - $B_\epsilon = \{\epsilon, a||b\}$
  - $B_a = \{a\}$
  - $B_b = \{b\}$
  - $B_1 = \{ab, aa, a||a, ...\}$

  $\}$

$\mathcal{D}$:

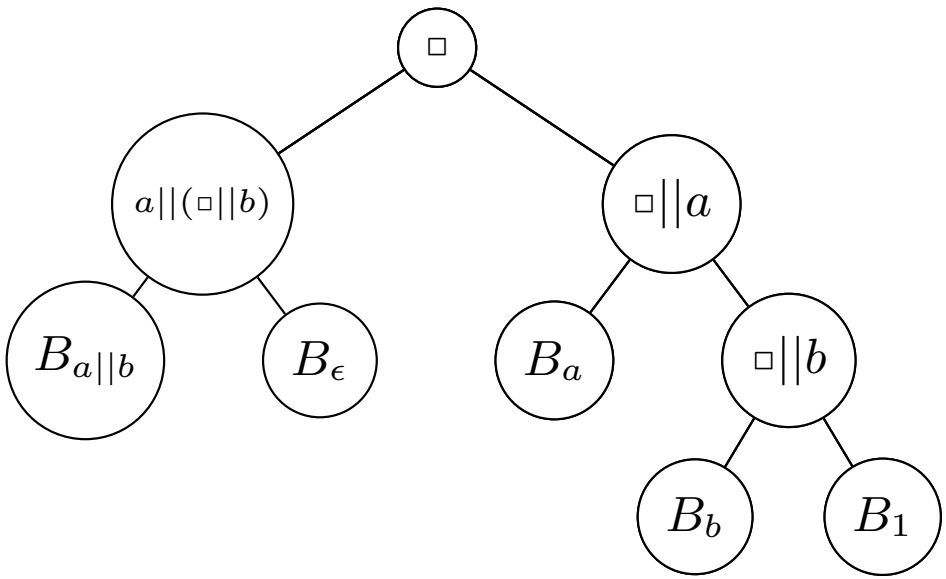


L^λ - exemple : Pb de consistance : Refine(B_ϵ,$a||(\square||b)$)

- $S = \{\epsilon, a, b, a||a\}$
- $S^+ = \{ab, aa, a(a||a), a(a||b), ...\}$
- $\mathcal{B} = \{$
  - $B_\epsilon = \{\epsilon\}$
  - $B_{a||b} = \{a||b, (a||b)(a||b), ...\}$
  - $B_a = \{a\}$
  - $B_b = \{b\}$
  - $B_1 = \{ab, aa, a||a, ...\}$

  $\}$

$\mathcal{D}$:

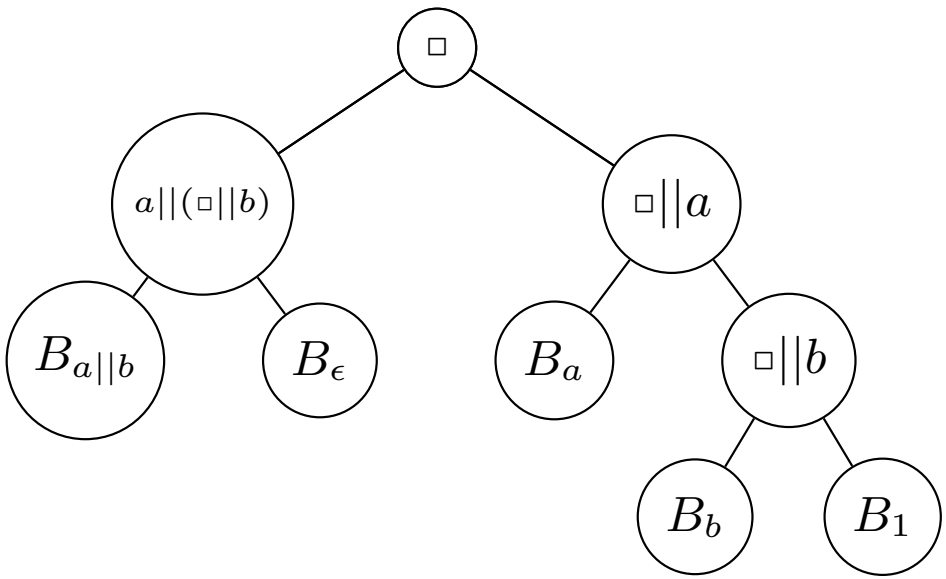


L^λ - exemple : Expand$(a||b)$

- $S = \{\epsilon, a, b, a||a, a||b\}$
- $S^+ = \{ab, aa, a(a||a), a(a||b), ...\}$
- $\mathcal{B} = \{$
  - $B_\epsilon = \{\epsilon\}$
  - $B_{a||b} = \{a||b, (a||b)(a||b), ...\}$
  - $B_a = \{a\}$
  - $B_b = \{b\}$
  - $B_1 = \{ab, aa, a||a, ...\}$

  $\}$

$\mathcal{D}$:



L^λ

- Produit un **PR minimal** depuis un **MAT** quelconque.

L^λ

- Produit un **PR minimal** depuis un **MAT** quelconque.

- **"Symbol complexity"** :

  - $L^\lambda : \mathcal{O}(2^n \cdot n^4 + k \cdot 2^n \cdot n^2 + d \cdot n^2 \cdot (2^n + m))$

  - $L^* : \mathcal{O}((2^n + m) \cdot n^3 + k \cdot m \cdot n + m \cdot n \cdot (2^n + m))$

L^λ

- Produit un **PR minimal** depuis un **MAT** quelconque.
- **"Symbol complexity"** :
  - $L^\lambda : \mathcal{O}(2^n \cdot n^4 + k \cdot 2^n \cdot n^2 + d \cdot n^2 \cdot (2^n + m))$
  - $L^* : \mathcal{O}((2^n + m) \cdot n^3 + k \cdot m \cdot n + m \cdot n \cdot (2^n + m))$
- Optimisation particulièrement intéressante dans le cas de grands contre-exemples.

Travail accompli - théorique

- Adapter L^λ aux **PR**.

Travail accompli - théorique

- Adapter L^λ aux **PR**.

- Adapter la décomposition de contre-exemple de **Rivest-Schapire** aux **PR** (donc aux arbres binaires).

Travail accompli - théorique

- Adapter L^λ aux **PR**.
- Adapter la décomposition de contre-exemple de **Rivest-Schapire** aux **PR** (donc aux arbres binaires).
- Proposer une méthode de comparaisons de **PR** qui permet de ne faire que des requêtes d'appartenance.

Travail accompli - théorique

- Adapter L^λ aux **PR**.
- Adapter la décomposition de contre-exemple de **Rivest-Schapire** aux **PR** (donc aux arbres binaires).
- Proposer une méthode de comparaisons de **PR** qui permet de ne faire que des requêtes d'appartenance.
- Montrer que notre version de L^λ conserve les avantages sur L^* qu'elle a dans le cas des mots.

Travail accompli - pratique

- Proposer des modélisations pratiques des **Pomsets** et des **PR**.

Travail accompli - pratique

- Proposer des modélisations pratiques des **Pomsets** et des **PR**.
- Proposer des implémentations des questions décidables sur les **PR**.

Travail accompli - pratique

- Proposer des modélisations pratiques des **Pomsets** et des **PR**.
- Proposer des implémentations des questions décidables sur les **PR**.
- Implémenter L^*, L^λ, la **W-method** et la décomposition de contre-exemple à la **Rivest-Schapire**.

Travail futur

- Génération de **PR** par système de réduction.

Travail futur

- Génération de **PR** par système de réduction.
- Créer un benchmark de **PR** pour comparer L^* et L^λ.

Travail futur

- Génération de **PR** par système de réduction.
- Créer un benchmark de **PR** pour comparer L^* et L^λ.
- Étendre ces méthodes aux automates d'arbre.

Publication

- Notre article est disponible sur ArXiv [6].

6. Pommellet, Amrane et Delaporte, *Active Learning Techniques for Pomset Recognizers*

Publication

- Notre article est disponible sur ArXiv [6].

- Nous soumettons à ICALP pour le 8 février.

6. Pommellet, Amrane et Delaporte, *Active Learning Techniques for Pomset Recognizers*


Amrane, Amazigh. « Posets série-parallèles transfinis : automates, logiques et théories équationnelles ». Theses. Normandie Université, oct. 2020. URL : https://theses.hal.science/tel-03215265.


Amrane, Amazigh et al. *Logic and Languages of Higher-Dimensional Automata*. 2024. arXiv : 2403.19526 [cs.FL].


Angluin, Dana. « Learning regular sets from queries and counterexamples ». In : *Information and Computation* 75.2 (1987), p. 87-106. ISSN : 0890-5401. DOI : https://doi.org/10.1016/0890-5401(87)90052-6. URL : https://www.sciencedirect.com/science/article/pii/0890540187900526.


Bedon, Nicolas. « Branching Automata and Pomset Automata ». In : *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*. Sous la dir. de Mikolaj Bojanczyk et Chandra Chekuri. T. 213. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 37 :1-37 :13. DOI : 10.4230/LIPICS.FSTTCS.2021.37. URL : https://doi.org/10.4230/LIPIcs.FSTTCS.2021.37.


Bloom, S.L. et Z. Ésik. « Free shuffle algebras in language varieties ». In : *Theoretical Computer Science* 163.1-2 (1996), p. 55-98.


Bruyère, Véronique, Olivier Carton et Géraud Sénizergues. « Tree automata and automata on linear orderings ». en. In : *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications* (2009). DOI : 10.1051/ita/2009009. URL : http://www.numdam.org/articles/10.1051/ita/2009009/.


Büchi, J.R. et D. Siefkes. *Finite Automata, Their Algebras and Grammars : Towards a Theory of Formal Expressions*. Springer New York, 1989. ISBN : 9783540969051. URL : https://books.google.fr/books?id=F9lQAAAAMAAJ.


Carton, Olivier. « Accessibility in Automata on Scattered Linear Orderings ». In : *Mathematical Foundations of Computer Science 2002*. Sous la dir. de Krzysztof Diks et Wojciech Rytter. Berlin, Heidelberg : Springer Berlin Heidelberg, 2002.

Chow, Tsun S. « Testing Software Design Modeled by Finite-State Machines ». In : *IEEE Trans. Software Eng.* 4.3 (1978), p. 178-187. DOI : 10.1109/TSE.1978.231496. URL : https://doi.org/10.1109/TSE.1978.231496.

Comon, Hubert et al. *Tree Automata Techniques and Applications*. 2008, p. 262. URL : https://inria.hal.science/hal-03367725.

Fahrenberg, Uli et al. « A Kleene Theorem for Higher-Dimensional Automata ». In : *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*. Sous la dir. de Bartek Klin, Slawomir Lasota et Anca Muscholl. T. 243. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 29 :1-29 :18. DOI : 10.4230/LIPICS.CONCUR.2022.29. URL : https://doi.org/10.4230/LIPIcs.CONCUR.2022.29.

Fahrenberg, Uli et al. « Languages of Higher-Dimensional Automata ». In : (2021). DOI : 10.48550/ARXIV.2103.07557. URL : https://arxiv.org/abs/2103.07557.

Fahrenberg, Uli et al. « Posets with interfaces as a model for concurrency ». In : *Inf. Comput.* 285.Part (2022), p. 104914. DOI : 10.1016/J.IC.2022.104914. URL : https://doi.org/10.1016/j.ic.2022.104914.

Fishburn, Peter C. « Interval graphs and interval orders ». In : *Discret. Math.* 55.2 (1985), p. 135-149. DOI : 10.1016/0012-365X(85)90042-1. URL : https://doi.org/10.1016/0012-365X(85)90042-1.

Fisun, Mykola, Hlib Horban et Kandyba Ihor. « Processing of Relational Algebra Expressions by the Shunting Yard Algorithm ». In : *2019 IEEE 14th International Conference on Computer Sciences and Information Technologies (CSIT)*. T. 3. 2019, p. 240-243. DOI : 10.1109/STC-CSIT.2019.8929740.

Floyd, Robert W. « Algorithm 97 : Shortest Path ». In : *Commun. ACM* 5.6 (1962), p. 345. ISSN : 0001-0782. DOI : 10.1145/367766.368168. URL : https://doi.org/10.1145/367766.368168.

Gischer, Jay L. « The Equational Theory of Pomsets ». In : 61 (1988), p. 199-224. DOI : 10.1016/0304-3975(88)90124-7. URL : http://dx.doi.org/10.1016/0304-3975(88)90124-7.

Grabowski, Jan. « On partial languages ». In : 4.2 (1981), p. 427.

Hanneforth, Thomas, Andreas Maletti et Daniel Quernheim. « Random Generation of Nondeterministic Finite-State Tree Automata ». In : *Electronic Proceedings in Theoretical Computer Science* 134 (nov. 2013), 11–16. ISSN : 2075-2180. DOI : 10.4204/eptcs.134.2. URL : http://dx.doi.org/10.4204/EPTCS.134.2.

Heerdt, Gerco van et al. « Learning Pomset Automata ». In : *CoRR* abs/2102.07504 (2021). arXiv : 2102.07504. URL : https://arxiv.org/abs/2102.07504.

Howar, Falk et Bernhard Steffen. « Active Automata Learning as Black-Box Search and Lazy Partition Refinement ». In : *A Journey from Process Algebra via Timed Automata to Model Learning - Essays Dedicated to Frits Vaandrager on the Occasion of His 60th Birthday*. Sous la dir. de Nils Jansen, Mariëlle Stoelinga et Petra van den Bos. T. 13560. Lecture Notes in Computer Science. Springer, 2022, p. 321-338. DOI : 10.1007/978-3-031-15629-8_17. URL : https://doi.org/10.1007/978-3-031-15629-8_17.

Ipate, Florentin et Logica Banica. « W-method for Hierarchical and Communicating Finite State Machines ». In : *2007 5th IEEE International Conference on Industrial Informatics*. T. 2. 2007, p. 891-896. DOI : 10.1109/INDIN.2007.4384891.

Isberner, Malte, Falk Howar et Bernhard Steffen. « The TTT Algorithm : A Redundancy-Free Approach to Active Automata Learning ». In : *Runtime Verification*. Sous la dir. de Borzoo Bonakdarpour et Scott A. Smolka. Cham : Springer International Publishing, 2014, p. 307-322. ISBN : 978-3-319-11164-3.

Jacobson, G. « Space-efficient static trees and graphs ». In : *30th Annual Symposium on Foundations of Computer Science*. 1989. DOI : 10.1109/SFCS.1989.63533.

Jones, Joel. « Abstract Syntax Tree Implementation Idioms ». In : *Pattern Languages of Program Design* (2003). Proceedings of the 10th Conference on Pattern Languages of Programs (PLoP2003) http ://hillside.net/plop/plop2003/papers.html. URL : http://hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf.

Kappé, Tobias et al. « On series-parallel pomset languages : Rationality, context-freeness and automata ». In : *Journal of Logical and Algebraic Methods in Programming* 103 (2019), p. 130-153. DOI : 10.1016/j.jlamp.2018.12.001. URL : https://doi.org/10.1016%2Fj.jlamp.2018.12.001.

Kearns, M. J. et U. V. Vazirani. *An Introduction to Computational Learning Theory*. Cambridge, MA, USA : MIT Press, 1994.

Knuth, D. E. et P. B. Bendix. « Simple Word Problems in Universal Algebras ». In : *Automation of Reasoning : 2 : Classical Papers on Computational Logic 1967–1970*. Sous la dir. de Jörg H. Siekmann et Graham Wrightson. Berlin, Heidelberg : Springer Berlin Heidelberg, 1983, p. 342-376. ISBN : 978-3-642-81955-1. DOI : 10.1007/978-3-642-81955-1_23. URL : https://doi.org/10.1007/978-3-642-81955-1_23.

Kuske, Dietrich. « Infinite Series-Parallel Posets : Logic, Algebra, Automata, and Languages ». In : (nov. 2000).

Le, Dan-Tuong. « Quantitative Analysis of Counterexample Generation for Automata Learning ». Thèse de doct. Universitätsbibliothek der RWTH Aachen, 2020.

Lodaya, K. et P. Weil. « A Kleene Iteration for Parallelism ». In : *Foundations of Software Technology and Theoretical Computer Science*. 1998, p. 355-366.

— .« Series-parallel posets : Algebra, automata and languages ». In : avr. 2006. ISBN : 978-3-540-64230-5. DOI : 10.1007/BFb0028590.

Lodaya, Kamal et Pascal Weil. « A Kleene Iteration for Parallelism ». In : *Foundations of Software Technology and Theoretical Computer Science*. 1998. URL : https://api.semanticscholar.org/CorpusID:16177533.


Lodaya, Kamal et Pascal Weil. « A Kleene Iteration for Parallelism ». In : *Foundations of Software Technology and Theoretical Computer Science, 18th Conference, Chennai, India, December 17-19, 1998, Proceedings*. Sous la dir. de Vikraman Arvind et Ramaswamy Ramanujam. T. 1530. Lecture Notes in Computer Science. Springer, 1998, p. 355-366. DOI : 10.1007/978-3-540-49382-2_33. URL : https://doi.org/10.1007/978-3-540-49382-2_33.


— .« Rationality in algebras with a series operation ». In : *Information and Computation* 171.2 (2001), p. 269-293.


— .« Series–parallel languages and the bounded-width property ». In : *Theoretical Computer Science* 237.1-2 (2000), p. 347-380.


Maranda, J.-M. « Formal Categories ». In : *Canadian Journal of Mathematics* 17 (1965), 758–801. DOI : 10.4153/CJM-1965-076-0.


Moerman, Joshua. « Nominal Techniques and Black Box Testing for Automata Learning ». Thèse de doct. Radboud University, 2019.


Munro, J. Ian et Patrick K. Nicholson. « Succinct Posets ». In : *CoRR* abs/1204.1957 (2012). arXiv : 1204.1957. URL : http://arxiv.org/abs/1204.1957.


Nerode, A. « Linear Automaton Transformations ». In : *Proceedings of the American Mathematical Society* 9.4 (1958), p. 541-544. ISSN : 00029939, 10886826. URL : http://www.jstor.org/stable/2033204 (visité le 04/06/2024).


Nicolas, Bedon. « Logic and Branching Automata ». In : *Logical Methods in Computer Science* Volume 11, Issue 4 (2015). DOI : 10.2168/lmcs-11(4:2)2015. URL : https://doi.org/10.2168%2Flmcs-11%284%3A2%292015.


Norvell, Theodore. *Parsing Expressions by Recursive Descent*. https://www.engr.mun.ca/~theo/Misc/exp_parsing.htm. 1999.


Pommellet, Adrien, Amazigh Amrane et Edgar Delaporte. *Active Learning Techniques for Pomset Recognizers*. 2025. arXiv : 2501.03914 [cs.FL]. URL : https://arxiv.org/abs/2501.03914.


Pratt, Vaughan. « Modelling Concurrency with Partial Orders ». In : *Internationaljournal of Parallel Programming* 15 (avr. 2000).


Rivest, Ronald L. et Robert E. Schapire. « Inference of Finite Automata Using Homing Sequences ». In : *Inf. Comput.* 103.2 (1993), p. 299-347. DOI : 10.1006/INCO.1993.1021. URL : https://doi.org/10.1006/inco.1993.1021.


Vaandrager, Frits et al. « A New Approach for Active Automata Learning Based on Apartness ». In : *Tools and Algorithms for the Construction and Analysis of Systems*. Sous la dir. de Dana Fisman et Grigore Rosu. Cham : Springer International Publishing, 2022, p. 223-243. ISBN : 978-3-030-99524-9.


Valdes, J., R. E. Tarjan et E. L. Lawler. « The recognition of series parallel digraphs ». In : *SIAM J. Comput.* 11 (1982), p. 298-313.


Wang, Liang, Xinhua Xu et Changjie Hu. « An overview of testing generation methods for protocol conformance testing ». In : *2013 IEEE International Conference on Signal Processing, Communication and Computing (ICSPCC 2013)*. 2013, p. 1-4. DOI : 10.1109/ICSPCC.2013.6663880.