INF 302 : Langages & Automates

Chapitre 4 : Algorithmes et problèmes de décision sur les AEFDs

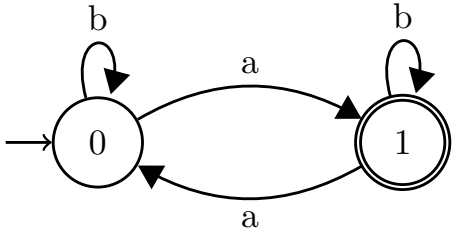
Yliès Falcone

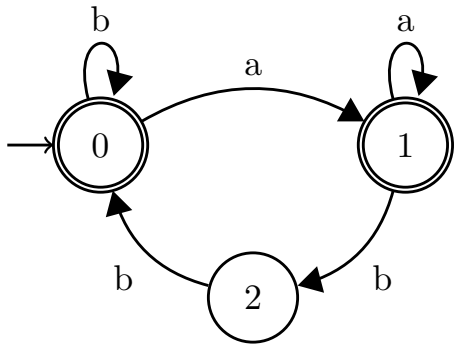
ylies.falcone@univ-grenoble-alpes.fr — www.ylies.fr

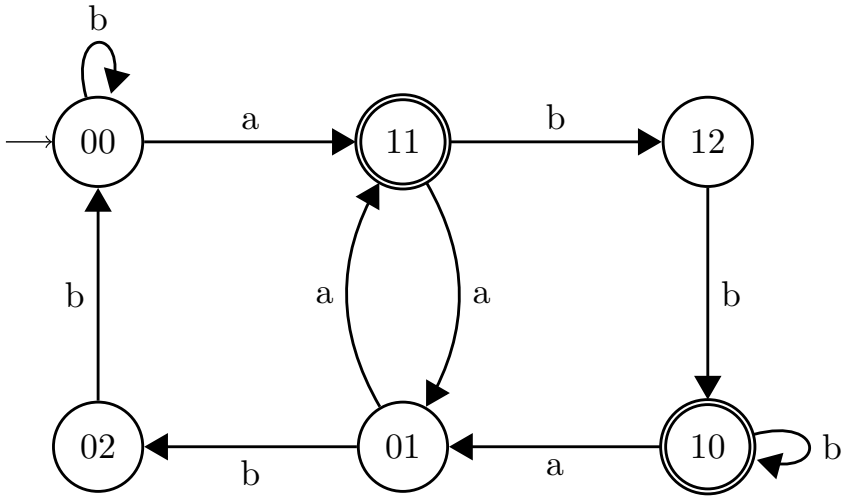
Univ. Grenoble-Alpes, Inria
-
Laboratoire d'Informatique de Grenoble - www.liglab.fr
Équipe de recherche LIG-Inria, Corse - team.inria.fr/corse/

Année Académique 2018 - 2019

Intuition et objectifs







- Parcours d'un automate : en largeur et profondeur.
- Détection de cycles.
- Notions d'accessibilité et de co-accessibilité.
- Problèmes de décision : langage vide, langage infini.

Plan Chap. 4 - Algorithmes et problèmes de décision sur les AEFDs

1. Parcours
2. Détection de cycles
3. Notions d'accessibilité et de co-accessibilité
4. Quelques problèmes de décision
5. Résumé

Plan Chap. 4 - Algorithmes et problèmes de décision sur les AEFDs

1. Parcours
2. Détection de cycles
3. Notions d'accessibilité et de co-accessibilité
4. Quelques problèmes de décision
5. Résumé

À propos du parcours d'un automate

Objectifs

- Visiter de manière *ordonnée* une seule fois chacun des états de l'automate (versus « pour tout état q dans Q . . . »).
- « brique de base » pour d'autres algorithmes sur les automates.

Principe

- suivi des transitions de l'automate,
- collecte d'information sur les états et transitions,
- propagation de propriétés.

Comment gérer les cycles ?

Algorithme paramétré par :

- une « opération de visite » des états de l'automate (où le traitement se fait) ;
- un « ordre » de parcours, influençant quels noeuds et quelles parties de l'automate sont visités en premier : profondeur d'abord ou largeur d'abord ;
- la « priorité » donnée entre les successeurs d'un état.

Notations Utiles pour le parcours d'automates

Dans la suite, nous utilisons un AEFD $(Q, \Sigma, q_{\text{init}}, \delta, F)$.

Successeurs d'un état

L'ensemble des états successeurs d'un état $q \in Q$, selon la fonction de transition δ, est :

$$\text{Succ}(q) = \left\{ q' \in Q \mid \exists a \in \Sigma : \delta(q, a) = q' \right\}$$

En itérant sur $\text{Succ}(q)$, nous obtenons les états selon la priorité donnée par les symboles.

Prédécesseurs d'un état

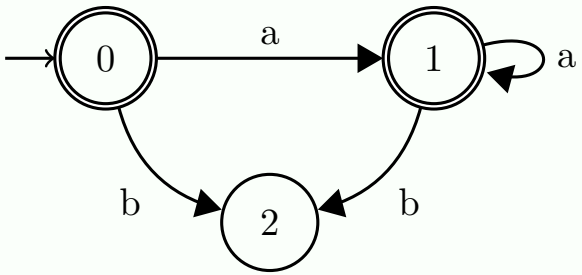
L'ensemble des états prédécesseurs d'un état $q \in Q$, selon la fonction de transition δ, est :

$$\text{Pré}(q) = \left\{ q' \in Q \mid \exists a \in \Sigma : \delta(q', a) = q \right\}$$

(En itérant sur $\text{Pré}(q)$, nous obtenons les états selon la priorité donnée par les symboles.)

Remarque Un état q est successeur d'un état q' ssi q' est un prédecesseur de q □

Exemple (Successeurs et prédécesseurs)

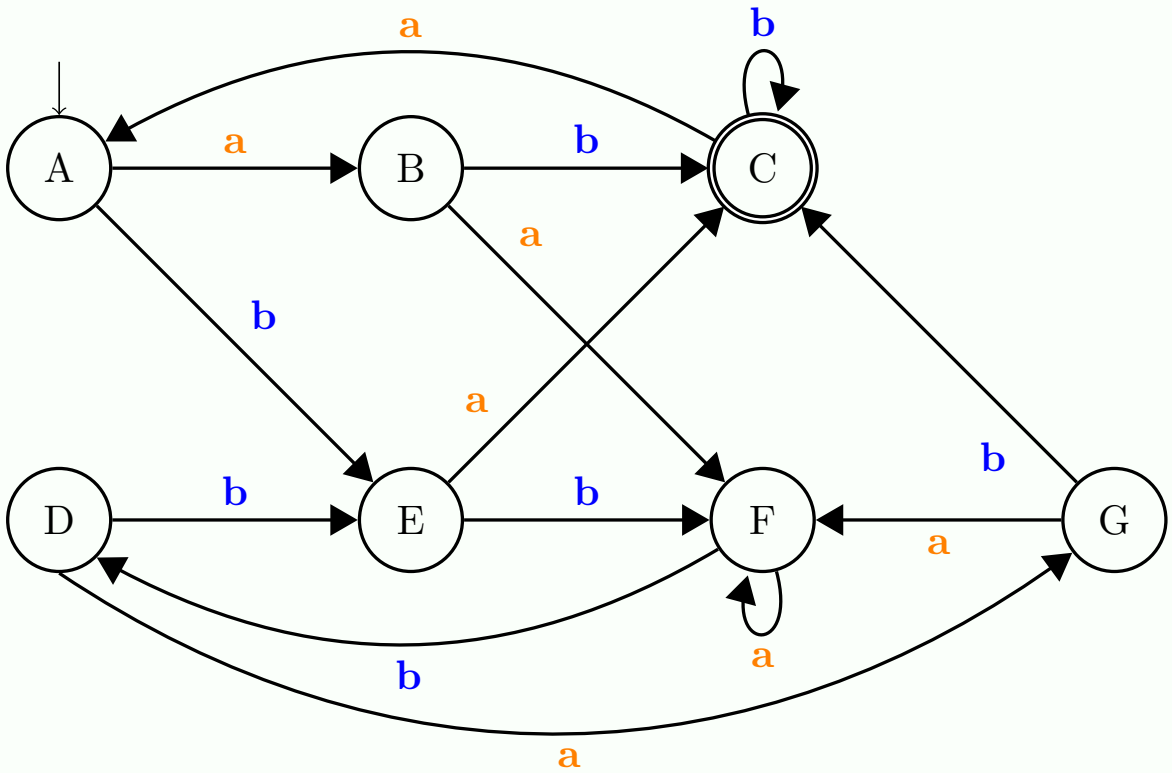


- successeurs de 0 : 1 et 2
- successeurs de 1 : 1 et 2
- successeurs de 2 : aucun

- prédécesseurs de 0 : aucun
- prédécesseurs de 1 : 0 et 1
- prédécesseurs de 2 : 0 et 2

Parcours en profondeur vs largeur d'abord depuis l'état initial

Exemple (Parcours en profondeur vs largeur d'abord)



◁ Parcours en prof. d'abord

◁ Parcours en largeur d'abord

Priorité de **a** sur **b**

- prof. d'abord : A, B, F, D, G, C, E
- largeur d'abord : A, B, E, F, C, D, G

Priorité de **b** sur **a**

- prof. d'abord : A, E, F, D, G, C, B
- largeur d'abord : A, E, B, F, C, D, G

Parcours d'un AEFD : version *récursive* en profondeur d'abord

Algorithme 1 *parcourir*() pour le parcours d'un automate

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ un AEFD

```
1: ens d'états Déjà_visité ;                    (* variable globale aux deux algorithmes *)
2: Déjà_visité := ∅ ;                           (* initialement, rien n'est visité *)
3: pour chaque état q dans l'ens. d'états de départ faire   (* graphe déconnecté *)
4:     si q ∉ Déjà_visité alors
5:         parcourir_rec(q)
6:     fin si
7: fin pour
8: retourner résultat ;
```

Algorithme 2 *parcourir_rec*() pour le parcours à partir de l'état q

Entrée : $q \in Q$ un état

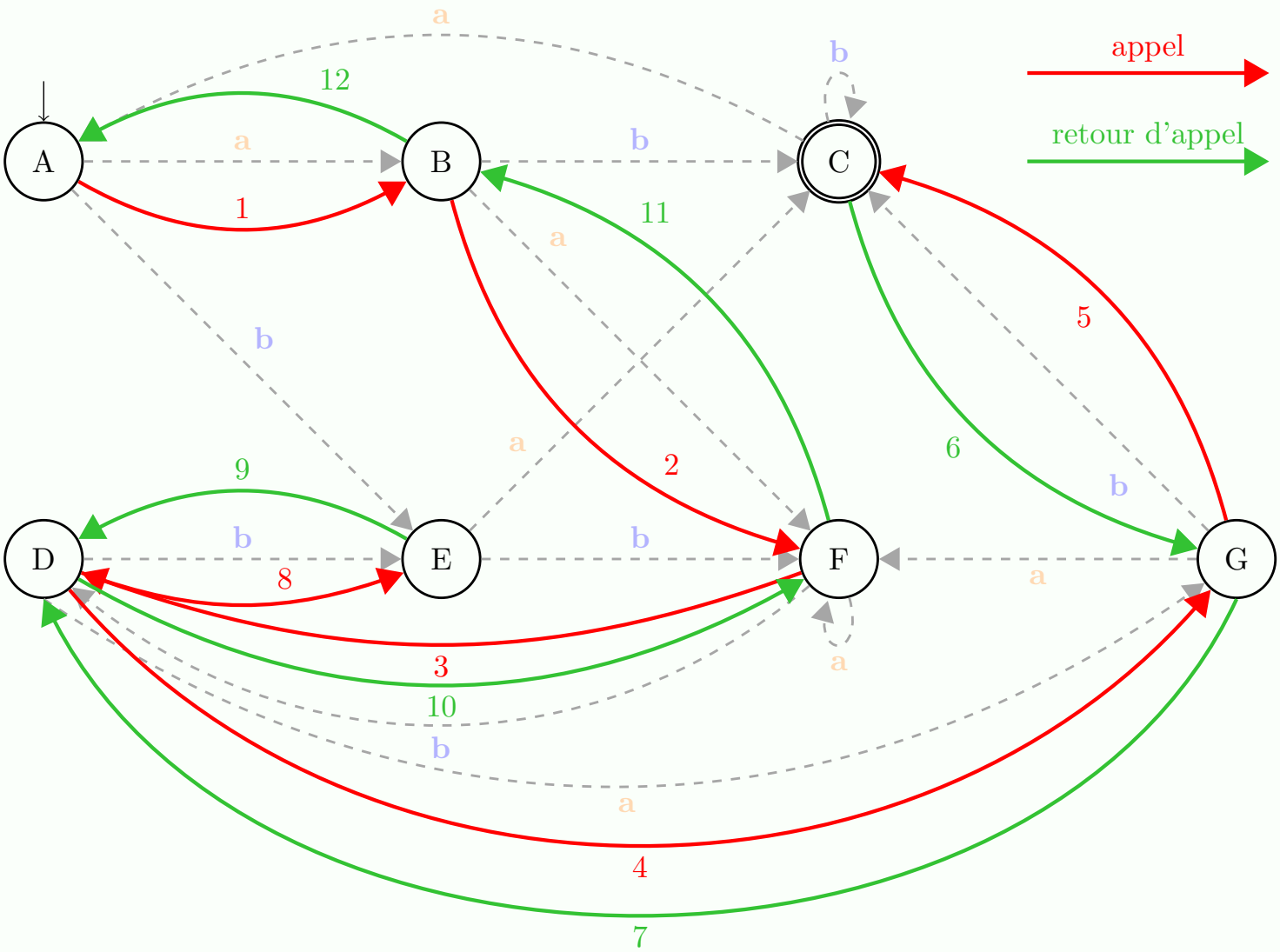
```
1: visiter l'état q ;
2: Déjà_visité := Déjà_visité ∪ {q} ;          (* l'état q n'est plus à visiter *)
3: pour chaque état q' ∈ Succ(q) \ Déjà_visité faire
4:     si q' ∉ Déjà_visité alors
5:         parcourir_rec(q')
6:     fin si
7: fin pour
```

Successeurs

À définir en fonction de l'algorithme basé sur le parcours. Par exemple, afficher.

Parcours d'un AEFD : version *récursive* en profondeur d'abord

Exemple (Parcours récursif en profondeur d'un AEFD)



Parcours d'un AEFD : version générique *itérative* de l'algorithme

Algorithme 3 *parcourir*() pour le parcours de l'automate

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ un AEFD

```
1:  ens d'états À_visiter := Ensemble d'états de départ de la visite ;
2:  ens d'états Déjà_visité := ∅ ;                    (* Initialement, rien n'est visité *)
3:  tant que À_visiter ≠ ∅ faire
4:      soit q ∈ À_visiter ;                          (* Prendre un état à visiter *)
5:      À_visiter := À_visiter \ {q} ;                (* L'état q n'est plus à visiter *)
6:      si q ∉ Déjà_visité alors
7:          Visiter l'état q ;
8:          Déjà_visité := Déjà_visité ∪ {q} ;        (* L'état q vient d'être visité *)
9:          À_visiter := À_visiter ∪ (Succ(q) \ Déjà_visité) ;
10:     fin si
11: fin tant que
12: retourner résultat ;
```

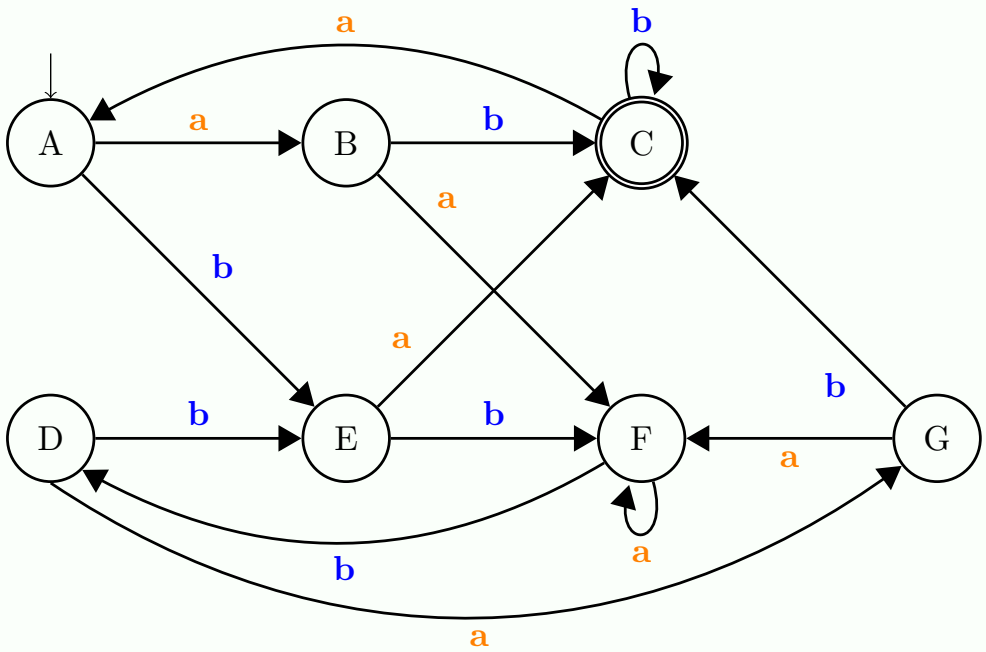
À raffiner en fonction de l'ordre de parcours :

- file : parcours en largeur d'abord
- pile : parcours en profondeur d'abord
- (file avec priorité : ≈Dijkstra, + court chemin)

À définir en fonction de l'algorithme basé sur le parcours.
P. ex. afficher l'état.

Parcours en largeur d'un AEFD : version *itérative* de l'algorithme

Exemple (Parcours itératif en largeur d'un AEFD)

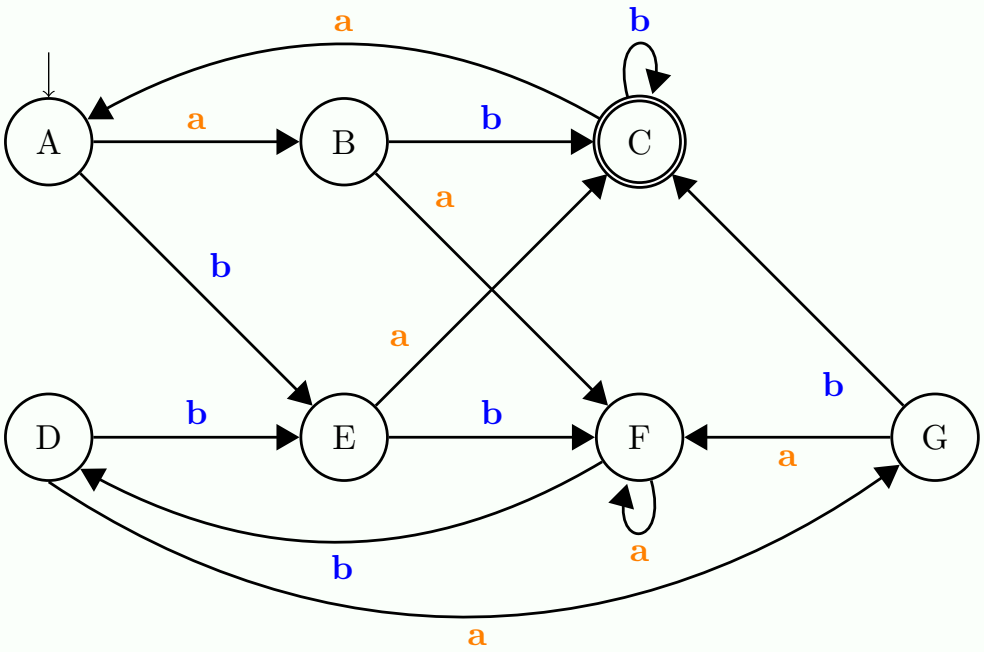


À visiter	Visités	Sortie
A		
E, B	A	A
C, F, E	A, B	B
C, F	A, B, E	E
D, C	A, B, E, F	F
D	A, B, C, E, F	C
G	A, . . . , F	D
	A, . . . , G	G

- À_visiter est une file, avec le début de file à gauche.
- Priorité de a sur b dans le choix des voisins.
  ↪ on enfile a puis b.

Parcours en profondeur d'un AEFD : version *itérative* de l'algorithme

Exemple (Parcours itératif en profondeur d'un AEFD)



À visiter	Visités	Sortie
A		
B, E	A	A
F, C, E	A, B	B
D, C, E	A, B, C	F
G, E, C, E	A, B, C, F	D
C, E, C, E	A, B, C, D, F	G
E, C, E	A, . . . , F	C
C, E	A, . . . , G	E
E	A, . . . , G	
	A, . . . , G	

- À_visiter est une pile, avec le haut de pile à gauche.
- Priorité de a sur b dans le choix des voisins.
  ↪ on empile b puis a.

Plan Chap. 4 - Algorithmes et problèmes de décision sur les AEFDs

1 Parcours

2 Détection de cycles

3 Notions d'accessibilité et de co-accessibilité

4 Quelques problèmes de décision

5 Résumé

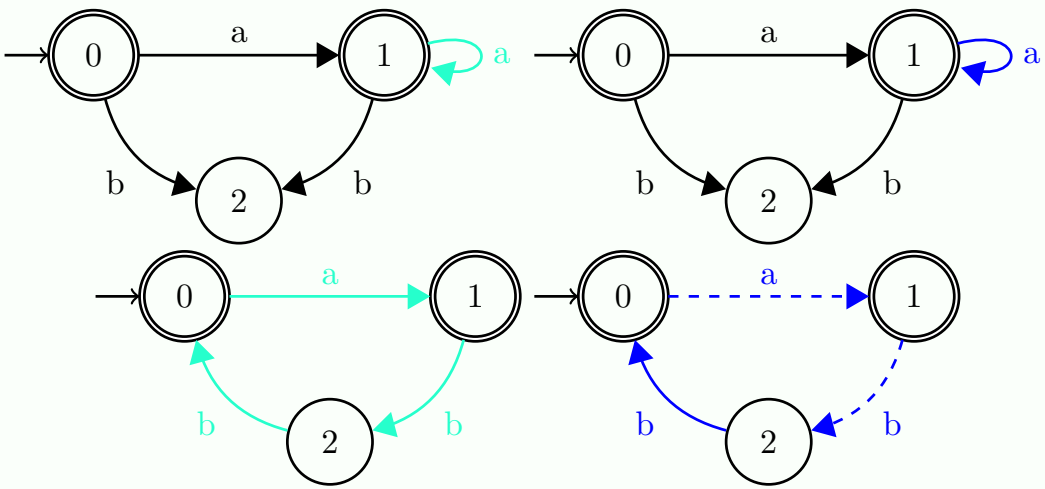
Detection de cycles

Définition (Cycle - deux définitions équivalentes)

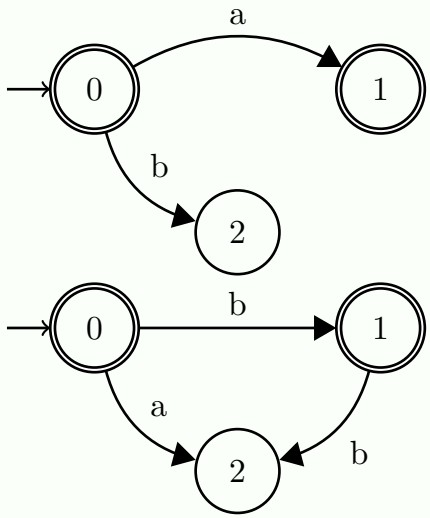
- séquence (non-vide) de transitions consécutives (l'état d'arrivée d'une transition est l'état de départ de la prochaine transition) t.q. le 1er et le dernier états soient identiques.
- automate avec une **transition arrière** :
  - soit une transition d'un état sur lui même,
  - soit une transition d'un état vers l'un de ces ancêtres dans l'arbre produit par le parcours en profondeur d'abord.

Exemple et contre-exemple (Cycle)

Automates avec cycle



Automates sans cycle

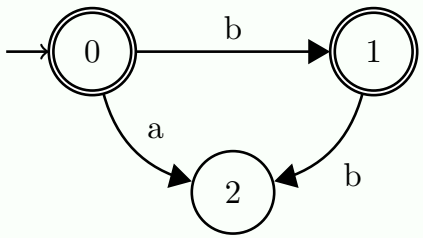


Detection de cycles - Intuition et remarque

Question

Lors d'un parcours en profondeur, est-ce que la découverte d'un état déjà visité implique l'existence d'un cycle ?

Exemple (Découvrir un état déjà visité n'implique pas l'existence de cycle)



Lors du parcours depuis l'état initial (où les voisins sont explorés dans l'ordre alphabétique) :
2 est visité, puis il est rencontré à nouveau.

Remarque La notion d'état visité n'est pas assez fine. Elle sert uniquement pour la terminaison de l'algorithme de parcours (et ne pas traiter des états déjà traités). □

Intuition pour trouver un cycle avec le parcours en profondeur récursif

- Se fait suivant la sous-structure d'arbre produit par le parcours.
- Lors du parcours d'origine un état q, il y a un cycle si on rencontre l'état q.
- **Se souvenir des noeuds par lesquels on est passé depuis et provenant de l'appel initial.**

◁ Détection de cycle avec parcours en profondeur.

Détection des cycles dans un AEFD

Basé sur l'algorithme récursif de parcours en profondeur d'abord - algorithme 1

Algorithme 4 *detection_cycle*() pour détecter l'existence d'un cycle dans un automate

Entrée : $A = (Q, \Sigma, \delta, q_{\text{init}}, F)$ un AEFD

1: **ens d'états** $\text{Déjà\_visité}, \text{Pile\_d\_appel} := \emptyset$;
2: **pour** chaque état $q \in Q$ **faire**
3: **si** *detection_cycle_état*(q) **alors retourner vrai fin si**
4: **fin pour retourner faux**

Algorithme 5 *detection_cycle_état*() pour détecter les cycles à partir d'un état

Entrée : $A = (Q, \Sigma, \delta, q_{\text{init}}, F)$ un AEFD, $q \in Q$ un état
Sortie : **vrai** s'il existe un cycle à partir de q, **faux** sinon

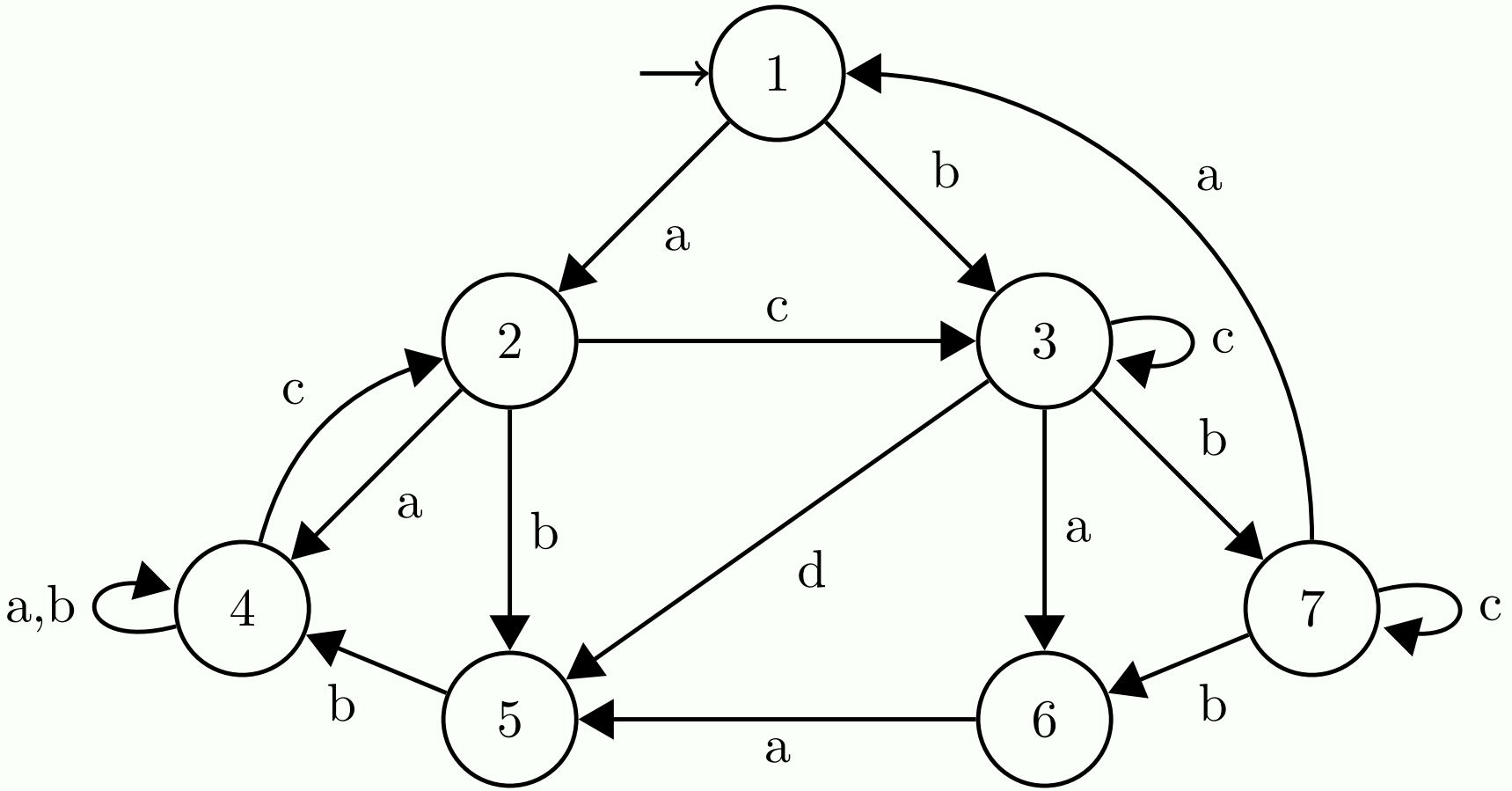
1: **si** $q \in \text{Pile\_d\_appel}$ **alors retourner vrai fin si**
2: **si** $q \in \text{Déjà\_visité}$ **alors retourner faux fin si**
3: $\text{Pile\_d\_appel} := \text{Pile\_d\_appel} \cup \{q\}$; (* On ajoute q à la pile d'appel *)
4: $\text{Déjà\_visité} := \text{Déjà\_visité} \cup \{q\}$; (* L'état q devient déjà visité *)
5: **pour** chaque état $q' \in \text{Succ}(q)$ **faire**
6: **si** *detection_cycle_état*(q') **alors retourner vrai fin si**
7: **fin pour**
8: $\text{Pile\_d\_appel} := \text{Pile\_d\_appel} \setminus \{q\}$; (* L'état q est supprimé de la pile d'appel *)
9: **retourner faux**

Remarque Cet algorithme s'adapte facilement pour retourner tous les cycles. □

Détection des cycles dans un AEFD

Basé sur l'algorithme récursif de parcours en profondeur d'abord - algorithme 1

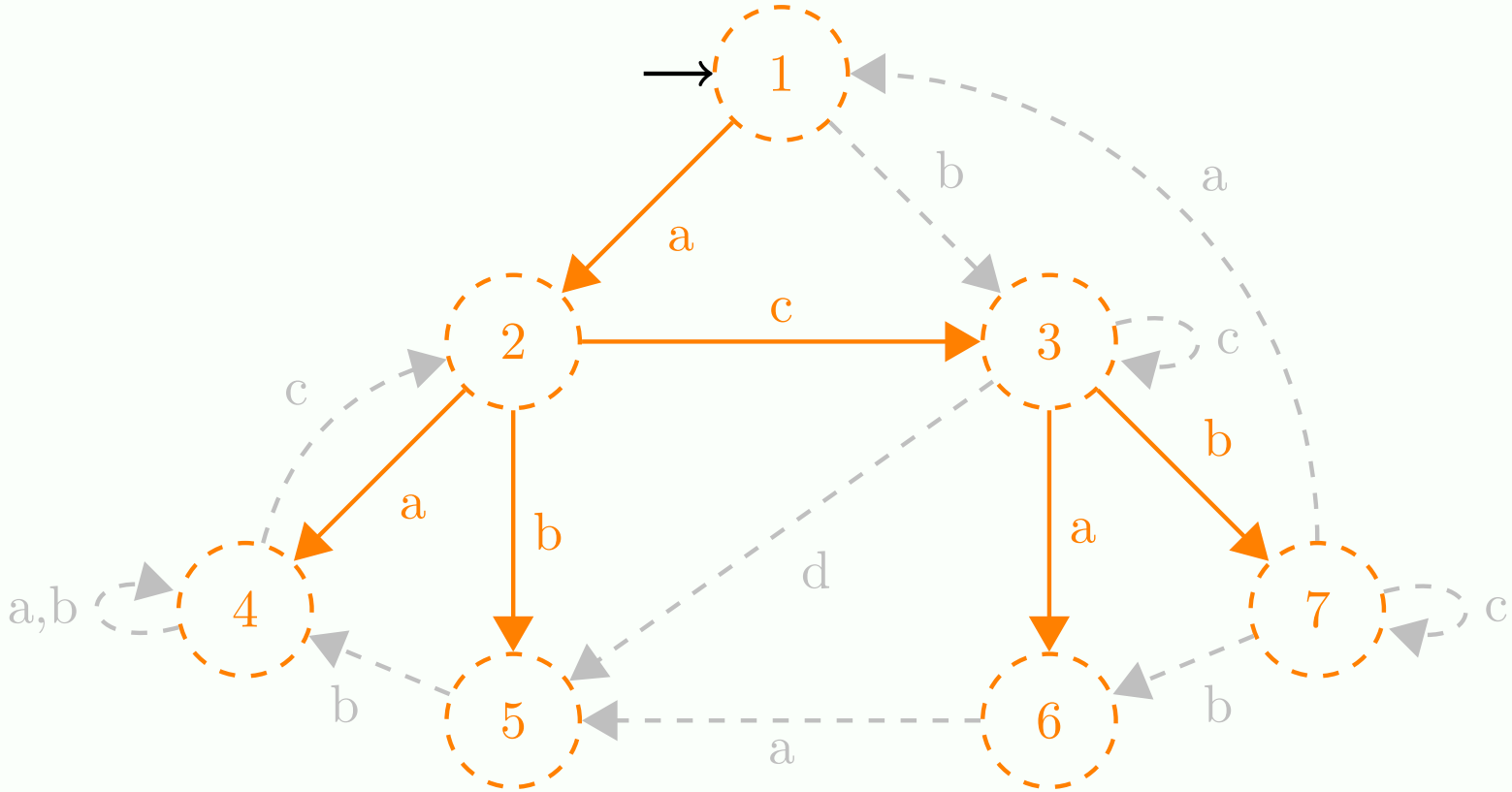
Exemple (Détection des cycles dans un AEFD avec l'algorithme 1)



Détection des cycles dans un AEFD

Basé sur l'algorithme récursif de parcours en profondeur d'abord - algorithme 1

Exemple (Détection des cycles dans un AEFD avec l'algorithme 1)



état dans l'arbre d'appel

état visité plus dans l'arbre d'appel

prochain état à visiter - non visité

prochain état à visiter - déjà visité

transition restant à explorer

transition élagée

prochaine transition à explorer

prochaine transition à explorer - non explorée

relation de parenté dans l'arbre produit par le parcours

transition arrière

Voir illustration animée correspondante

Détection des cycles dans un AEFD

Limites de l'algorithme 1 - Vers un algorithme 2

Limites de l'algorithme 1

- Les tests $q \in \text{Pile\_d\_appel}$ et $q \in \text{Déjà\_visité}$ sont en $\mathcal{O}(|Q|)$.
- La pile d'appel existe déjà grâce aux appels récursifs dans le parcours en profondeur.

Modifications à l'algorithme 1

- Utiliser un coloriage des états dans $\{\text{BLANC}, \textbf{GRIS}, \textbf{NOIR}\}$:
  - BLANC : état non traité ;
  - **GRIS** : état en cours de traitement, successeurs pas tous traités ;
    $\hookrightarrow$ l'état est dans la pile
  - **NOIR** : état et tous ses successeurs traités ;
    $\hookrightarrow$ l'état n'est dans la pile
- Tester la couleur d'un état est en $\mathcal{O}(1)$.

Détection des cycles dans un AEFD

Basé sur l'algorithme récursif de parcours en profondeur d'abord - algorithme 2

Algorithme 6 *detection_cycle*() pour détecter les cycles dans un AEFD

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ un AEFD, $q \in Q$ un état
Sortie : **vrai** s'il existe un cycle à partir de q, **faux** sinon

1: $\text{couleur} : Q \to \{\text{BLANC}, \textbf{GRIS}, \textbf{NOIR}\}$ (* Initialement : $\forall q \in Q : \text{couleur}(q) = \text{BLANC}$ *)
2: **pour** chaque état $q \in Q$ **faire**
3: **si** $\text{couleur}(q) == \text{BLANC}$ **alors** *detection_cycle_état*(q) **fin si**
4: **fin pour**

Algorithme 7 *detection_cycle_état*() pour détecter les cycles à partir d'un état

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ un AEFD, $q \in Q$ un état
Sortie : **vrai** s'il existe un cycle à partir de q, **faux** sinon

1: $\text{couleur}(q) := \textbf{GRIS}$
2: **pour** chaque état $q' \in \text{Succ}(q)$ **faire**
3: **si** $\text{couleur}(q') == \textbf{GRIS}$ **alors retourner vrai fin si**
4: **si** $\text{couleur}(q') = \text{BLANC}$ et *detection_cycle_état*(q') **alors retourner vrai**
(* q' n'a pas été traité et il y a une transition arrière dans le sous-arbre de racine q' *)
5: **fin si**
6: **fin pour**
7: $\text{couleur}(q) := \textbf{NOIR}$
8: **retourner faux**

Plan Chap. 4 - Algorithmes et problèmes de décision sur les AEFDs

1 Parcours

2 Détection de cycles

3 Notions d'accessibilité et de co-accessibilité
- Accessibilité
- Co-accessibilité
- Vocabulaire et utilisation

4 Quelques problèmes de décision

5 Résumé

Accessibilité et co-accessibilité

Notions liées à la fonction de transition des AEFDs.

Accessibilité et co-accessibilité : définition informelle

Propriétés s'appliquant aux états d'un AEFD :

- état accessible : peut être atteint à en suivant la fonction de transition ;
- état co-accessible : mène à un état accepteur en suivant la fonction de transition.

Nous utiliserons ces notions :

- pour répondre à des problèmes de décision sur les automates dans la section suivante ;
- caractériser certaines de leur propriétés en TD ;
- dans les chapitres suivants.

Plan Chap. 4 - Algorithmes et problèmes de décision sur les AEFDs

1 Parcours

2 Détection de cycles

3 Notions d'accessibilité et de co-accessibilité
- Accessibilité
- Co-accessibilité
- Vocabulaire et utilisation

4 Quelques problèmes de décision

5 Résumé

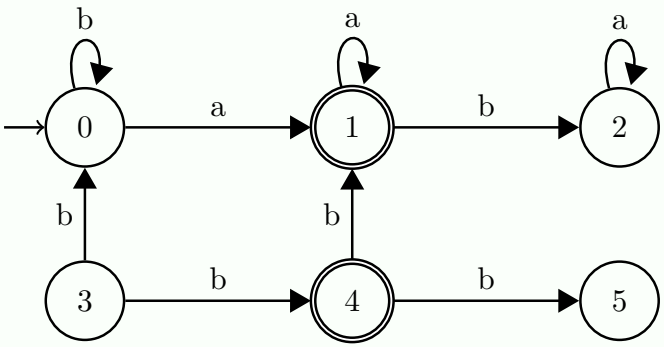
Accessibilité dans les AEFDs : définition et décidabilité

Etant donné un AEFD $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$.

Définition (Accessibilité d'un état dans un AEFD)

$q \in Q$ est accessible s'il existe un mot $u \in \Sigma^*$ tel que $\delta^*(q_{\text{init}}, u) = q$.

Exemple (États accessibles)



- accessibles : 0, 1, 2
- non accessibles : 3, 4, 5

Théorème : **accessibilité** dans les graphes finis

Le problème de *l'accessibilité est décidable* pour les graphes finis.

Corollaire

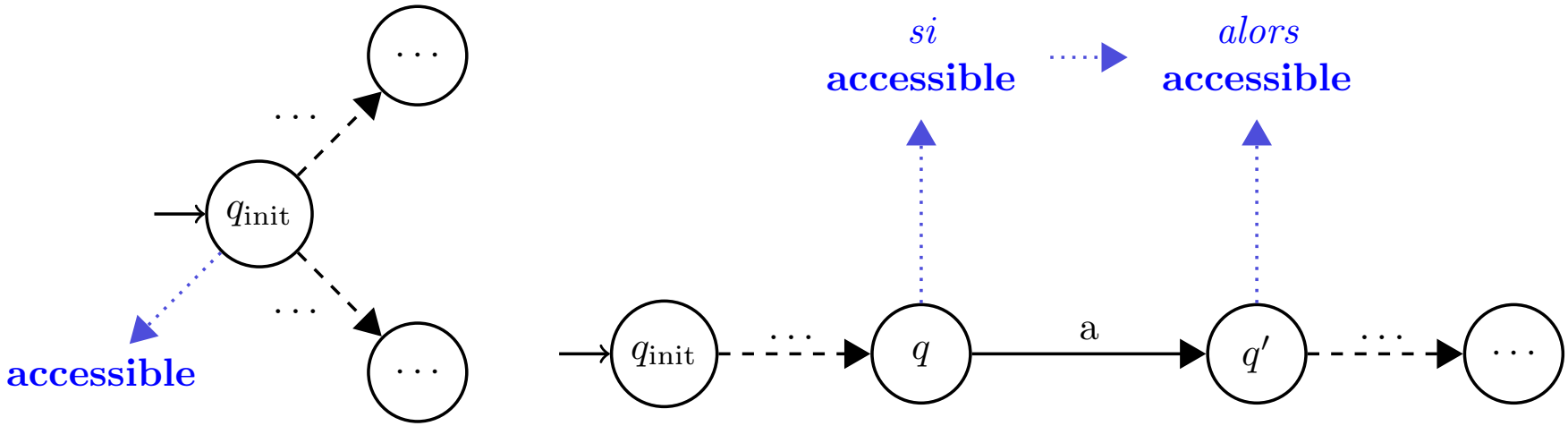
Le problème de l'accessibilité d'un état dans un AEFD est décidable.

Accessibilité dans les AEFDs : intuition sur l'algorithme

Etant donné un AEFD $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$.

Idée de l'algorithme

- case de base : q_{init} est accessible
- cas d'induction : si $q \in Q$ est accessible dans A, alors s'il existe une transition dans δ de q vers un état q', alors q' est accessible dans A ($q' \in \text{Succ}(q)$).



Accessibilité dans les AEFDs : algorithme itératif

Algorithme 8 *etats_accessibles*() pour le calcul des états accessibles

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ un AEFD
Sortie : $\text{Accessibles} \subseteq Q$ ensemble des états accessibles dans A par δ

1: **ens d'états** $\text{Accessibles}, \text{À\_visiter}, \text{Déjà\_visité}, R_{local}$;
2: $\text{Accessibles} := \{q_{\text{init}}\}$; (* Cas de base *)
3: $\text{À\_visiter} := \{q_{\text{init}}\}$; (* Parcours depuis l'état initial *)
4: $\text{Déjà\_visité} := \emptyset$; (* Initialement, rien n'est visité *)
5: **tant que** $\text{À\_visiter} \neq \emptyset$ **faire**
6: soit $q \in \text{À\_visiter}$;
7: $\text{À\_visiter} := \text{À\_visiter} \setminus \{q\}$;
8: $\text{Déjà\_visité} := \text{Déjà\_visité} \cup \{q\}$;
9: $\text{Accessibles} := \text{Accessibles} \cup \text{Succ}(q)$;
(* On ajoute les états accessibles depuis q à l'ensemble global des états accessibles *)
10: $\text{À\_visiter} := \text{À\_visiter} \cup \big(\text{Succ}(q) \setminus \text{Déjà\_visité}\big)$;
(* Les nouveaux états à visiter sont ceux « découverts » *)
11: **fin tant que**
12: **retourner** Accessibles ;

Remarque Cet algorithme s'adapte pour calculer les états accessibles à partir d'un état ou d'un ensemble d'états donnés de l'automate (en modifiant les lignes 2 et 3). □

Plan Chap. 4 - Algorithmes et problèmes de décision sur les AEFDs

1 Parcours

2 Détection de cycles

3 Notions d'accessibilité et de co-accessibilité
- Accessibilité
- Co-accessibilité
- Vocabulaire et utilisation

4 Quelques problèmes de décision

5 Résumé

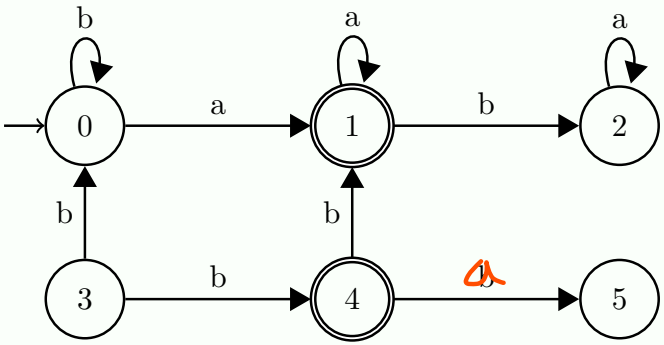
Co-accessibilité dans les AEFDs : définition et décidabilité

Etant donné un AEFD $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$.

Définition (Co-accessibilité d'un état dans un AEFD)

$q \in Q$ est co-accessible s'il existe un mot $u \in \Sigma^*$ tel que $\delta^*(q, u) \in F$.

Exemple (États co-accessibles)



- co-accessibles : 0, 1, 3, 4
- non co-accessibles : 2, 5

Théorème : **co-accessibilité** dans les graphes finis

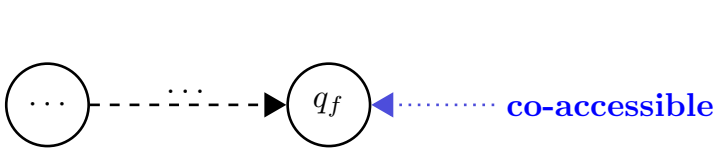
Le problème de *la co-accessibilité est décidable* pour les graphes finis.

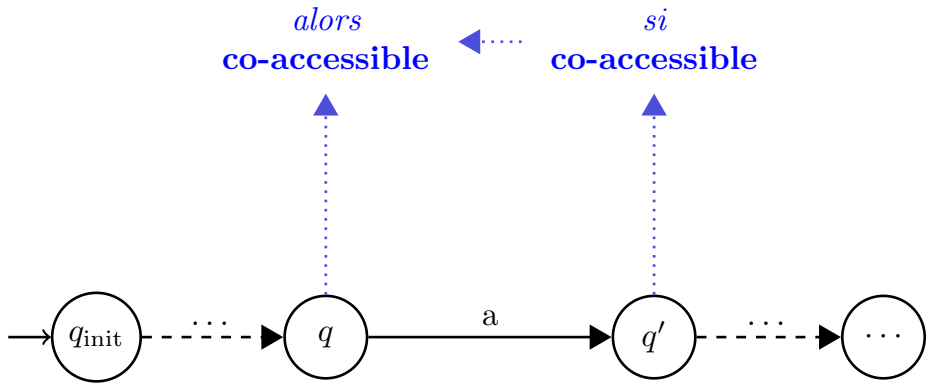
Co-accessibilité dans les AEFDs : intuition sur l'algorithme

Etant donné un AEFD $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$.

Définition (Idée de l'algorithme)

- case de base : les états $q \in F$ sont co-accessibles,
- cas d'induction : si $q \in Q$ est co-accessible dans A, alors s'il existe une transition dans δ depuis un état q' vers q, alors q' est co-accessible dans A.





Dans la suite, nous utilisons la notation suivante :

$$\text{Pré}(q) := \{q' \in Q \mid \exists a \in \Sigma : (q', a, q) \in \delta\}$$

pour les prédécesseurs de l'état q suivant la fonction de transition δ.

Remarque La co-accessibilité peut aussi se calculer facilement avec l'accessibilité entre deux états q, q' quelconques. □

Co-accessibilité dans les AEFDs : algorithme

Algorithme 9 *etats_coaccessibles*() pour le calcul des états accessibles

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ un AEFD
Sortie : $\text{Coaccessibles} \subseteq Q$ ensemble des états accessibles dans A par δ

1: **ens d'états** Coaccessibles, A_visiter, Déjà_visité, coR_{local} ;
2: $\text{Coaccessibles} := F$; (* Cas de base *)
3: $\text{A\_visiter} := F$; (* On veut les états co-accessibles à tous les états accesseurs *)
4: $\text{Déjà\_visité} := \emptyset$;
5: **tant que** $\text{A\_visiter} \neq \emptyset$ **faire**
6: soit $q \in \text{A\_visiter}$;
7: $\text{A\_visiter} := \text{A\_visiter} \setminus \{q\}$;
8: $\text{Déjà\_visité} := \text{Déjà\_visité} \cup \{q\}$;
9: $\text{Coaccessibles} := \text{Coaccessibles} \cup \text{Pré}(q)$;
10: $\text{A\_visiter} := \text{A\_visiter} \cup \big(\text{Pré}(q) \setminus \text{Déjà\_visité}\big)$;
11: **fin tant que**
12: **retourner** Coaccessibles ;

Cet algorithme peut facilement être modifié pour calculer les états co-accessibles à ensemble d'états donnés de l'automate (en modifiant les lignes 2 et 3).

Co-accessibilité en réutilisant l'accessibilité

La co-accessibilité peut se résoudre en ré-utilisant l'algorithme d'accessibilité, puis

- en ‘« inversant » la fonction de transition,
- en « partant des états accepteurs ».

Algorithme 10 *etats_coaccessibles*() pour le calcul des états accessibles

---

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ un AEFD
Sortie : ensemble des états co-accessibles dans A par δ

1: $\delta' = \{(q', a, q) \mid (q, a, q') \in \delta\} \cup \{(q_{\text{new}}, a, q_f) \mid q_f \in F \wedge a \in \Sigma\}$;
2: **automate** $E := (Q \cup \{q_{\text{new}}\}, \Sigma, \delta', q_{\text{new}}, q_{\text{init}})$;
3: **retourner** *etats_acessibles*$(E) \setminus \{q_{\text{new}}\}$;

Plan Chap. 4 - Algorithmes et problèmes de décision sur les AEFDs

1 Parcours

2 Détection de cycles

3 Notions d'accessibilité et de co-accessibilité
- Accessibilité
- Co-accessibilité
- Vocabulaire et utilisation

4 Quelques problèmes de décision

5 Résumé

Vocabulaire

Un automate dont tous les états sont accessibles est dit *accessible*.

Un automate dont tous les états sont co-accessibles est dit *co-accessible*.

Un automate accessible et co-accessible est dit **émondé**.

Intuitivement, dans un automate émondé, tous les états sont « utiles » à la reconnaissance des mots.

Utilisation

- Problèmes de décision (voir section suivante).
- Garder uniquement les états « utiles » d'un automate.
- Connaitre et générer les états d'un automate dans les situations suivantes :
  - Lorsqu'ils ne sont pas donnés de manière explicite, mais par exemple sous forme de règles. Exemple : automate de l'étrange planète vu en introduction du cours.
  - Lorsqu'on ne veut pas les générer a priori. Exemple : produit de deux automates.

En TD

- Garder uniquement les états accessibles d'un automate : étant donné un automate, écrire un algorithme qui retourne un automate équivalent avec tous ses états accessibles.
- Même question avec la co-accessibilité.
- Produire la version émondée d'un automate.
- Produit de deux automates « à la volée ».

Plan Chap. 4 - Algorithmes et problèmes de décision sur les AEFDs

1 Parcours

2 Détection de cycles

3 Notions d'accessibilité et de co-accessibilité

4 Quelques problèmes de décision

5 Résumé

Problèmes de décision
Définition

Définition (Problème de décision)

- Question que l'on peut *exprimer mathématiquement* (formellement).
- Question à un certain nombre de paramètres que l'on souhaite pouvoir passer à un programme informatique.
- La réponse à la question est *oui ou non*.

Deux sortes de problèmes de décision existent :

- **problèmes décidables** : on peut trouver un algorithme ou un programme qui sait répondre à la question (avec une mémoire et un temps non-borné).
- **problèmes indécidables** : on ne peut pas trouver un algorithme ou un programme qui sait répondre à la question (de manière générale).

Exemple (Problème de décision)

- décidables : évaluation d'un circuit, voyageur de commerce, SAT(isfiabilité), ...
- indécidables : arrêt d'une machine de Turing ou de Minsky (automate avec deux compteurs) ou d'un programme, solutions entières d'une équation Diophantienne, billet d'avion le moins cher entre deux villes, ...

Deux **problèmes de décision** sur les AEFDs

Considérons un AEFD A.

Problème du langage vide

Le langage reconnu par A est il vide ?

Problème de la finitude du langage

Le langage reconnu par A est-il (de cardinalité) fini(e) ?

Pour répondre à ces questions, nous allons utiliser les notions d'*accessibilité*, de *co-accessibilité* et de *cycle*.

Décision du problème du **langage vide**

Théorème : décision du problème du langage vide

Le problème du *langage vide est décidable* pour les automates à états finis (déterministes).

Algorithme 11 Procédure de décision pour le problème du langage vide

Entrée : $A = (Q, \Sigma, \delta, q_{\text{init}}, F)$ un AEFD

Sortie : **vrai** si le langage reconnu par A est vide, **faux** sinon

1: **ens d'états** $\text{Accessibles} := \textit{etats\_accessibles}(A)$;

2: **retourner** $\big(\text{Accessibles} \cap F\big) == \emptyset$;

Décision du problème de **la finitude du langage**

Théorème : décision du problème de **la finitude du langage**

Le problème de *la finitude du langage est décidable* pour les AEFDs.

Idée intuitive de l'algorithme

- Le langage accepté ne sera pas fini, si l'automate peut accepter "autant de mots qu'on veut".
- Il y a un nombre fini d'états.
- Il faut pouvoir donc arriver dans un état accepteur par un nombre infini de chemins différents.
- Il faut l'existence d'un cycle.

Le langage reconnu par un automate $(Q, \Sigma, \delta, q_{\text{init}}, F)$ est infini ssi :
il existe un cycle non trivial accessible et co-accessible.

Décision du problème de **la finitude du langage**

Algorithme 12 Procédure de décision pour le problème du langage infini

Entrée : $A = (Q, \Sigma, \delta, q_{\text{init}}, F)$ un AEFD

Sortie : “vrai" si le langage reconnu par A est infini, "faux" sinon

1: **ens d'états** $\text{EtatsEmonde} := \textit{etats\_accessibles}(A) \cap \textit{etats\_coaccessibles}(A)$;

2: **automate** $E := (Q \cap \text{EtatsEmonde}, \Sigma, \delta \cap \text{EtatsEmonde} \times \Sigma \times \text{EtatsEmonde}, q_{\text{init}}, F \cap \text{EtatsEmonde})$;

3: **retourner** $\{q \in \text{EtatsEmonde} \mid \textit{detection\_cycle\_util}(E, q)\} \neq \emptyset$;

Plan Chap. 4 - Algorithmes et problèmes de décision sur les AEFDs

1 Parcours

2 Détection de cycles

3 Notions d'accessibilité et de co-accessibilité

4 Quelques problèmes de décision

5 Résumé

Résumé du Chap. 4 - Algorithmes et problèmes de décision sur les AEFDs

Automate à États Fini Déterministe

- Parcours :
  - profondeur et largeur d'abord,
  - récursif vs itératif.
- Détection de cycle dans un automate :
  - basée sur un parcours en profondeur d'abord,
  - détection de transition arrière,
- Notions d'accessibilité et de co-accessibilité :
  - état accessible : existence d'un chemin depuis l'état initial vers cet état,
  - état co-accessible : existence d'un chemin depuis cet état jusqu'à un état accepteur.
- *Décidabilité* de certains *problèmes de décision* :
  - langage vide,
  - finitude du langage.

Remarque Les notions des prochains chapitres et les exercices de TD nous permettront de répondre à d'autre problèmes de décision. □

Remarque Les algorithmes de parcours, détection de cycle, de calcul des états accessibles et co-accessibles restent valables pour les automates non-déterministes que nous aborderons dans un prochain chapitre. □