Langages réguliers & Expressions régulières

Les langages réguliers jouent un grand rôle dans la théorie des langages et la compilation. Ils sont liés à deux notions :

- les expressions régulières, de nature algébrique :
  - qui permettent de les définir (on dit qu'elles dénotent ces langages),
  - qui sont également utilisées pour définir les unités lexicales d'un langage de programmation
- les automates finis, de nature géométrique, qui permettent de les reconnaître.

Dans toute la suite, les alphabets sont supposés finis

Langages réguliers & Expressions régulières

Définition

- Tout langage fini peut être énuméré (même si cela peut être long).
- Pour les langages infinis une énumération exhaustive n'est pas possible
- La classe des langages réguliers comprend tous les langages finis et certains langages infinis

Définition

- Langages réguliers ⇒ Expressions régulières
- Expressions régulières ⇒ Opérations régulières
- ???

$\Sigma = \{a, b\}$

$L = \{ u \in \Sigma^* \}$

$(a+b)^*$

DBeaver CE
DataGrip

Langages réguliers & Expressions régulières

Opérations régulières

Sont ainsi appelées les trois opérations suivantes définies sur les langages :

- réunion finie
- concaténation
- itération

Langages réguliers :

Un langage est dit régulier si et seulement si on peut le construire, à partir de langages finis par un nombre fini d'applications d'opérations régulières.

Langage régulier = interprétation d'une expression régulière

$\Sigma = \{a, b\}$

$L = \{ u \in \Sigma^* \}$

$(0+1)^*$

DBeaver CE
DataGrip

Langages réguliers & Expressions régulières

La notion d'expression régulière est d'usage fréquent en informatique. On y a souvent recours pour :

- rechercher certains motifs récurrents
- sélectionner des fichiers en fonction de critères plus fins.

répertoire contenant divers fichiers

```
> ls monrepertoire/
memoire.aux   memoire.tex      picture001.jpg      rapsody.jpg
memoire.dvi   picture001.jpg   presentation.exe    raw.jpg
memoire.old   picture002.jpg   price-list.txt
memoire.log   picture003.jpg   taches.txt
```

- Si l'utilisateur désire afficher uniquement les images au format "JPEG" et comportant l'extension .jpg $\Longrightarrow$ ls *.jpg
- De la même manière, s'il veut effacer tous les fichiers relatifs à memoire, il exécutera $\Longrightarrow$ rm m*

$\varepsilon^* \{$

DBeaver CE
DataGrip

Langages réguliers & Expressions régulières

Expression régulière

$ERreg(\Sigma)$ souvent noté aussi $\mathcal{R}_\Sigma$ est l'ensemble des expressions définies par application des clauses inductives :

a/ $x \in \mathcal{R}_\Sigma$ pour tout $x \in \Sigma$

b/ $\emptyset \in \mathcal{R}_\Sigma$

c/ Si $\alpha \in \mathcal{R}_\Sigma$ et $\beta \in \mathcal{R}_\Sigma$, alors $(\alpha + \beta) \in \mathcal{R}_\Sigma$

d/ Si $\alpha \in \mathcal{R}_\Sigma$ et $\beta \in \mathcal{R}_\Sigma$, alors $(\alpha.\beta) \in \mathcal{R}_\Sigma$

e/ Si $\alpha \in \mathcal{R}_\Sigma$ alors $(\alpha^*) \in \mathcal{R}_\Sigma$

Quelques expressions régulières pour $\Sigma = \{a, b\}$

a/ $\alpha_1 = \epsilon + (a.b)$

b/ $\alpha_2 = (((a.b).a) + b^*)$

c/ $\alpha_3 = ((a + b)^*.(a.b))$

DBeaver CE
DataGrip

Langages réguliers & Expressions régulières

Interprétation d'expressions régulières

La première application de ce principe d'induction est la définition d'une interprétation $I : \mathcal{R}_\Sigma \to \mathcal{P}(\Sigma^*)$ des expressions régulières par des langages. I est définie par induction :

a/ $I(x) = x$ pour tout $x \in \Sigma$ (langage dont le seul élément est le mot à une seule lettre x)

b/ $I(\emptyset) = \emptyset$ (langage vide)

c/ $I(\alpha + \beta) = I(\alpha) + I(\beta)$ (réunion de langages)

d/ $I(\alpha.\beta) = I(\alpha).I(\beta)$ (concaténation de langages)

e/ $I(\alpha^*) = I(\alpha)^*$ (itération d'un langage)

Autre notation

On utilise également la notation 2^{Σ^*} pour désigner $\mathcal{P}(\Sigma^*)$

$u \in \Sigma^* \}$

DBeaver CE
DataGrip

Interprétation d'expressions régulières

Quelques interprétations d'expressions régulières pour $\Sigma = \{a, b\}$

a/ $\alpha_1 = \epsilon + (a.b) \Longrightarrow I(\alpha_1) = \{\epsilon, ab\}$

b/ $\alpha_2 = (((a.b).a) + b^*) \Longrightarrow I(\alpha_1) = (\{aba\} \cup \{b\}^*)^*$

c/ $\alpha_3 = ((a+b)^*.(a.b)) \Longrightarrow I(\alpha_1) = \{a, b\}^*.\{ab\}$

Définition

- Un langage L sur Σ est régulier s'il existe une expression régulière Φ telle que

$$L = I(\Phi)$$

- Si ϕ et ψ sont deux expressions régulières telles que $I(\phi)$ et $I(\psi)$, on dit alors que ϕ et ψ sont équivalents.

DBeaver CE
DataGrip

Expressions régulières Vs Interprétation d'expressions régulières

Il y a peu de différence entre une expression régulière et son interprétation : il n'est pas très dangereux de les confondre. Par exemple :

- $(\alpha + \beta)^* = (\alpha^* + \beta^*)^*$ veut dire $(L + M)^* = (L^* + M^*)^*$ où $L = I(\alpha)$ et $M = I(\beta)$ $+M)^* = (L^* + M^*)^*$
- De même, on écrit $(ab + a^*)ba$ au lieu de $(((((a.b) + a^*)).b).a)$ ou $((((a.b) + a^*)).(b.a))$

Conclusion

Une expression régulière est, littéralement, une présentation formelle d'un langage régulier. Elle définit la forme (ou le motif) qui sert de modèle aux mots du langage en question.

Convention d'écriture

Dans la suite, on s'autorisera à confondre une expression régulière et le langage qu'elle représente. Si aucune confusion n'est possible, on s'autorisera également à enlever les parenthèses ou autres symboles superflus

Quelques exemples de langages réguliers

- Le langage vide $\emptyset \subseteq \Sigma^*$.
- Le langage $\epsilon \subseteq \Sigma^*$ car $l(\emptyset^* = \epsilon)$.
- Un langage réduit à un seul mot est régulier. En particulier tout $x \in \Sigma$ est un langage régulier.
- Tout langage fini.
- $\Sigma \subseteq \Sigma^*$.
- En appliquant la clôture de $\mathcal{R}_\Sigma$ par itération, on déduit du point précédent que $\Sigma^* \subseteq \Sigma^*$ est régulier.

- Nous avons vu (Cf. lemme d'Arden) que la plus petite solution d'un système d'équations linéaires se calcule à partir de ses coefficients, par application d'opérations régulières : or, ces opérations transforment des langages réguliers en des langages réguliers. On a donc :

Propriété

La plus petite solution d'un système d'équations linéaires à coefficients réguliers est constituée de langages réguliers.

- De plus, on peut remarquer que le calcul de cette solution (par la "méthode de Gauss") est effectif, lorsque le système est fini.
- Un cas important est celui o'u les coefficients sont des langages finis

DBeaver CE
DataGrip

Les automates finis : Motivations

Motivations

On utilise les automates non seulement pour savoir

- Ce qu'on peut et ce qu'on ne peut pas décrire avec un automate.
- Définir des opération pour composer des comportements.
- Développer des algorithmes sur les automates.

Mais aussi pour caractériser les problèmes qu'on peut résoudre à l'aide d'une "machine à calculer". Autrement dit :

- Existe-t-il une limite à ce qu'on peut programmer ?
- Quelle est cette limite si elle existe ?

$\varepsilon^* \{$

DBeaver CE
DataGrip

Existe-t-il une limite à ce qu'on peut programmer ?

Un des évènements clé c'est l'invention de la Machine de Turing : la "machine idéale" décrite par Alan Turing en 1936. C'est une machine abstraite qui n'est rien d'autre qu'un automate.

Quelques réponses

- Oui il y a des problèmes qu'on ne peut pas résoudre à l'aide d'un ordinateur même si on suppose une mémoire non-bornée (illimitée).
- Cette limite est caratérisée de plusieurs manières équivalentes :
  - Automates.
  - Classes de fonctions.
  - Logiques

$\Sigma = \{a, b\}$

$L = \{ u \in \Sigma^* \}$

$(0+1)^*$

$a^n b^n$

DBeaver CE

DataGrip

Les automates

Définition littérale & Concepts

- Un automate fini est une machine abstraite constituée d'états et de transitions.
- Cette machine est destinée à traiter des mots fournis en entrée : l'automate passe d'état en état, suivant les transitions, à la lecture de chaque lettre de l'entrée.
- L'automate est dit "fini" car il possède un nombre fini d'états distincts : il ne dispose donc que d'une mémoire bornée, indépendamment de la taille de la donnée sur laquelle on effectue les calculs.

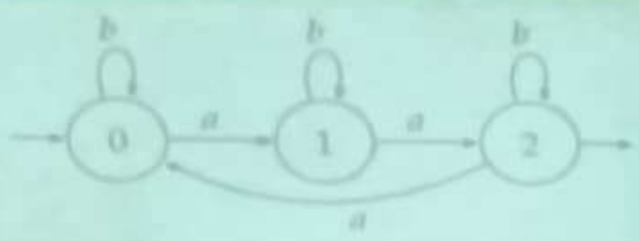
DBeaver CE
DataGrip

Les automates

Définition littérale & Concepts

- Un automate est constitué d'états et de transitions. Son comportement est dirigé par un mot fourni en entrée : l'automate passe d'état en état, suivant les transitions, à la lecture de chaque lettre de l'entrée.
- On distingue différentes façons de représenter un automate.

Représentation graphique Vs Représentation tabulaire



δ	a	b
0	1	0
1	2	1
2	0	2

$L = \{ u \in \Sigma^* \}$

$(0+1)^*$

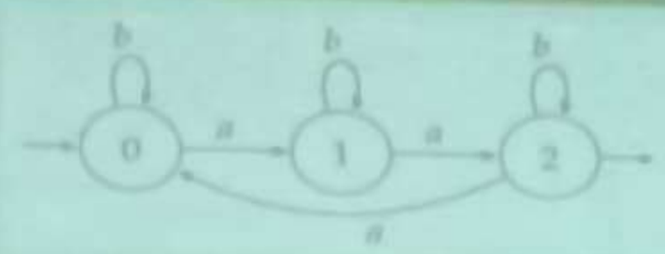
$a^n b^n$

DBeaver CE

DataGrip

Les automates

Représentation graphique Vs Représentation tabulaire



δ	a	b
0	1	0
1	2	1
2	0	2

Dans l'exemple ci-dessus, pour l'entrée bbaabaaabb, et si l'automate démarre en 0, il passe successivement par les états $0 \to 0 \to 0 \to 1 \to 2 \to 2 \to 0 \to 1 \to 2 \to 2 \to 2 \to$.

- Un tel automate est dit "fini" car il possède un nombre fini d'états : il ne dispose donc que d'une mémoire bornée.

$\{ u \in \Sigma^* \}$

$(\ldots)^*$

$a^n b^n$

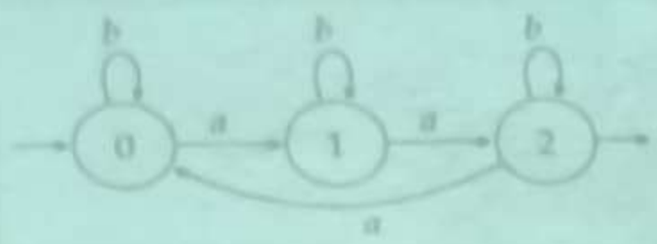
DBeaver CE
DataGrip

Les automates

Definition littérale & Concepts

- On peut très bien considérer des automates sans limitation sur le nombre d'états : la théorie qui en résulte est très analogue à la théorie habituelle.

Représentation graphique Vs Représentation tabulaire



δ	a	b
0	1	0
1	2	1
2	0	2

L'état initial est marqué par une flèche entrante ; un état final est, selon les auteurs, soit marqué d'une flèche sortante, comme sur la figure ci-dessus, soit doublement cerclé.

$u \in \Sigma^* \}$

$(a)^*$

$a^n b^n$

DBeaver CE

DataGrip

Différents types d'automates

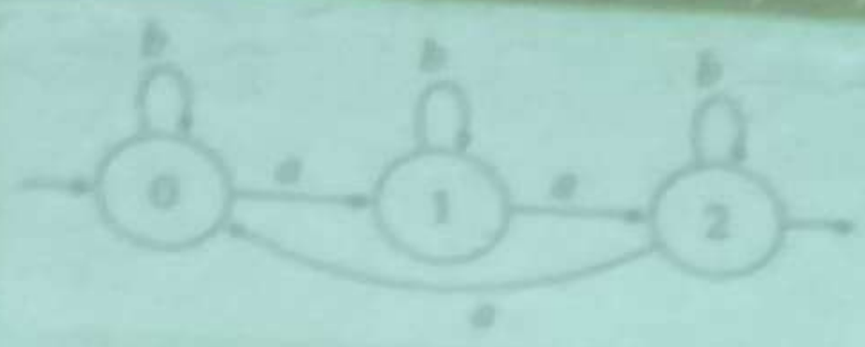
Dans cette section, nous introduisons le modèle le plus simple d'automate fini : l'automate fini déterministe complet (AFDC).

Automate Fini Déterministe Complet (AFDC)

Un automate fini déterministe (complet) (DFA) est défini par un quintuplet $A = (\Sigma; Q; q_0; F; \delta)$, où :

- Σ est un ensemble fini de symboles (l'alphabet)
- Q est un ensemble fini d'états
- $q_0 \in Q$ est l'état initial
- $F \subset Q$ est l'ensemble des états finaux
- δ est une fonction totale de $Q \times \Sigma$ dans Q, appelée fonction de transition.

Automate fini déterministe (complet)



δ	a	b
0	1	0
1	2	1
2	0	2

- Un automate fini correspond à un graphe orienté, dans lequel certains des noeuds (états) sont distingués et marqués comme initial ou finaux et dans lequel les arcs (transitions) sont étiquetés par des symboles de Σ.
- Une transition est donc un triplet de $Q \times \Sigma \times Q$. Si $\delta(q, a) = r$ on dit que a est l'étiquette de la transition (q, a, r) ; q en est l'origine, et r la destination.
- La fonction de transition correspondant à ce graphe s'exprime matriciellement par le graphe ci-dessus

DBeaver CE
DataGrip

Automate fini déterministe (complet)

- Un calcul dans l'automate A est une séquence de transitions $e_1 \ldots e_n$ de A, telle que pour tout couple de transitions sucessives e_i, e_{i+1}, l'état destination de e_i est l'état origine de e_{i+1}.
- L'étiquette d'un calcul est le mot construit par concaténation des étiquettes de chacune des transitions.
- Un calcul dans A est réussi si la première transition a pour état d'origine l'état initial, et si la dernière transition a pour destination un des états finaux.
- Le langage reconnu par l'automate A, noté L(A), est l'ensemble des étiquettes des calculs réussis.

D

ans l'exemple précédent, le mot baab appartient au langage reconnu, puisqu'il étiquette le calcul (réussi) :
$(0, b, 0), (0, a, 1), (1, a, 2), (2, b, 2)$.

Automate fini déterministe (complet)

- La relation $\vdash_A$ permet de formaliser la notion d'étape élémentaire de calcul. Ainsi on écrira, pour a dans Σ et v dans Σ^* $(q, av) \vdash_A (\delta(q, a), v)$ pour noter une étape de calcul utilisant la transition $(q, a, \delta(q, a))$, ce qui correspond à $(q \bullet av) = (q \bullet a) \bullet v$ vu plus haut.
- La clôture réflexive et transitive de $\vdash_A$ se note $\vdash_A^*$; ainsi, $(q, uv) \vdash_A^* (p, v)$ s'il existe une suite d'états $q = q_1 \ldots q_n = p$ tels que $(q1, u1 \ldots u_n v) \vdash_A (q2; u2 \ldots u_n v) \ldots \vdash_A (q_n; v)$.
- La réflexivité de cette relation signifie que pour tout (q, u), $(q, u) \vdash_A^* (q, u)$. Avec ces notations, on a :

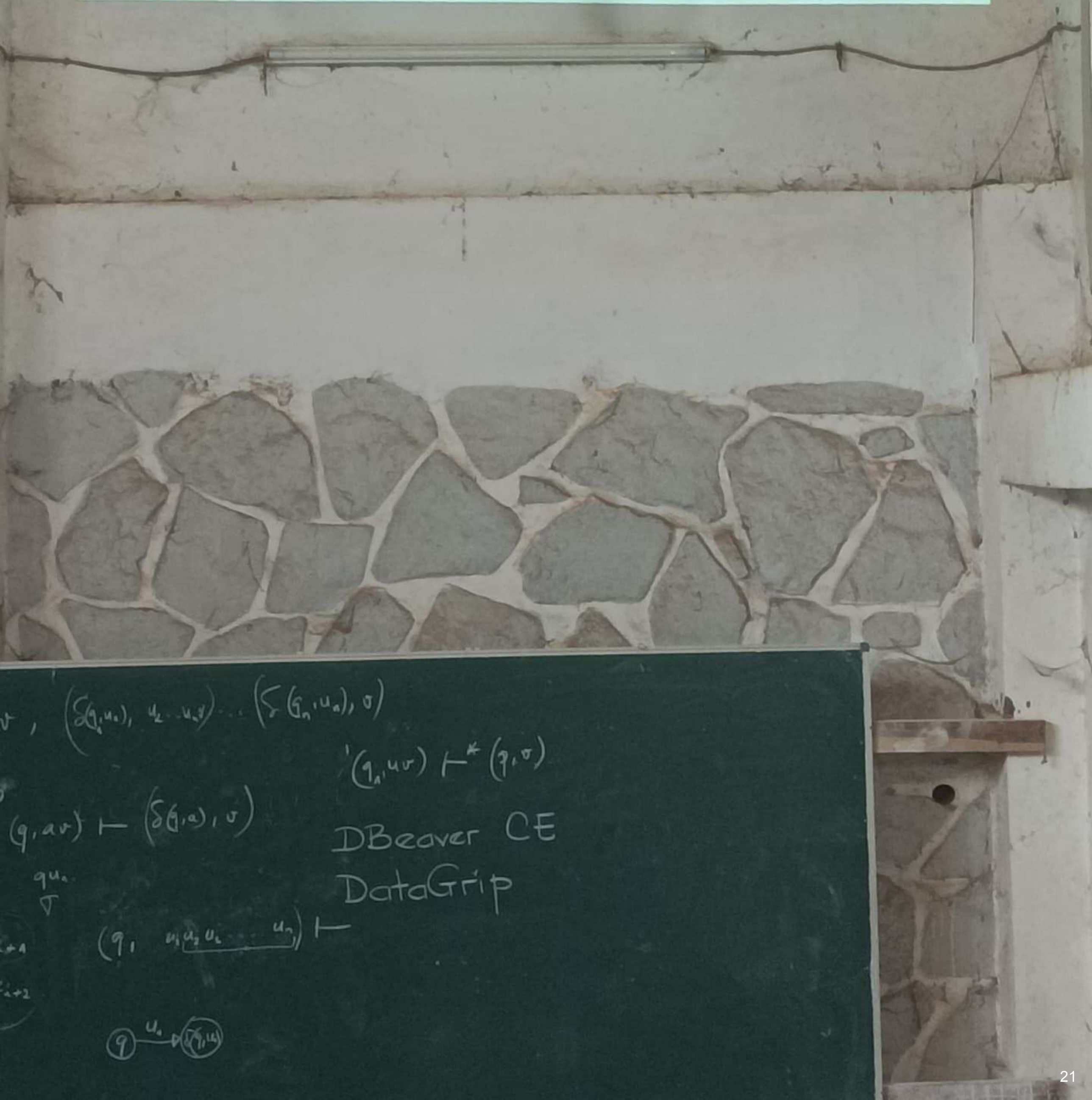
$$L(A) = \{u \in \Sigma^* \mid (q_0, u) \vdash_A^* (q, \epsilon), \text{ avec } q \in F\}$$

DBeaver CE
DataGrip

Automate fini déterministe (complet)

$$L(A) = \{u \in \Sigma^* \mid (q_0, u) \vdash_A^* (q, \epsilon), \text{ avec } q \in F\}$$

- Automate vu comme un Reconnaisseur (parcours des symboles du mot un à un). On dérive un algorithme permettant de tester si un mot appartient au langage reconnu par un automate fini déterministe.
- Chaque étape de calcul correspond à une application de la fonction $\delta()$, qui elle-même se réduit à la lecture d'une case d'un tableau et une affectation, deux opérations qui s'effectuent en temps constant.
- La reconnaissance d'un mot u se calcule en exactement $|u|$ étapes.



Algo de reconnaissance d'un mot par un AFD

Automate fini déterministe (complet)

On peut donc écrire un algorithme permettant de reconnaître un DFA :

Algorithme

// $u = u_1 \dots u_n$ est le mot à reconnaître
// $A = (Q, q0, \delta, F)$ est le DFA
$q := q_0$
$i := 1$
<u>while</u>$(i \leq n)$ **<u>do</u>**
$q := \delta(q, u_i)$
$i := i + 1$
od
<u>if</u>$(q \in F)$ **<u>then</u>** *return(true)* **<u>else</u>** *return(false)*

$(\delta(q_1, u_1), u_2 \dots u_n v) \dots (\delta(q_n, u_n), v)$

$(q_1, uv) \vdash^* (q, v)$

$(q, av) \vdash (\delta(q, a), v)$

qu_1
v

$(q, u_1 u_2 u_3 \dots u_n) \vdash$

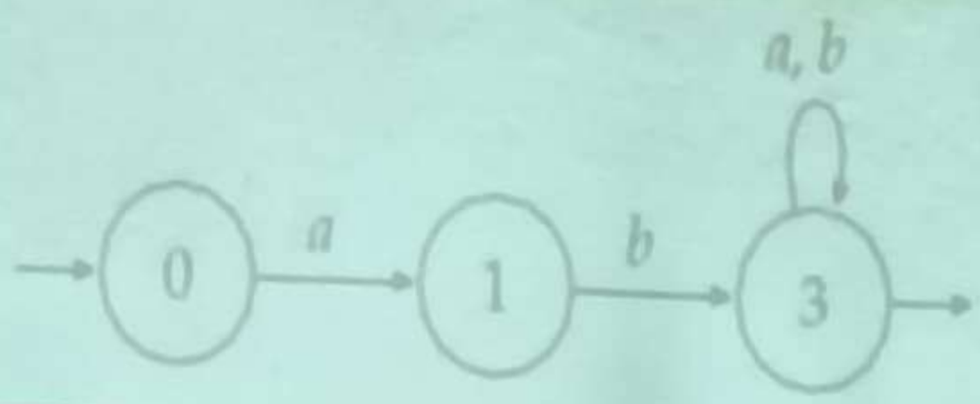
DBeaver CE
DataGrip

Automate fini déterministe

Lors de l'arrêt, la machine est dans l'une des trois configurations suivantes :

- Le mot n'est pas entièrement lu mais aucune transition n'est possible : le mot est rejeté. aa
- Le mot a été lu, mais l'état dans lequel se trouve la machine n'est pas un état final : le mot est rejeté. a
- Le mot a été lu, et l'état dans lequel se trouve la machine est un état final : le mot est accepté. abab

Automate fini déterministe



DBeaver CE

DataGrip

Automate fini déterministe

- Si la fonction de transition δ est une application (est totale), on a un AFD Complet
- Si la fonction de transition δ est une fonction partielle, on parle simplement d'AFD.
- La différence avec la définition de l'AFDC est que δ est ici définie comme une fonction partielle. Son domaine de définition est un sous-ensemble de $Q \times \Sigma$. Selon cette nouvelle définition, il est possible de se trouver dans une situation où un calcul s'arrête avant d'avoir atteint la fin de l'entrée.

AFD Vs AFDC

Il existe une stricte équivalence entre AFD et AFDC.

DBeaver CE

DataGrip

Transformation d'un AFD en AFDC

AFD —> AFDC

- Ajout d'un état puits
- Absorption par l'état puits de toutes les transitions absentes de l'automate original

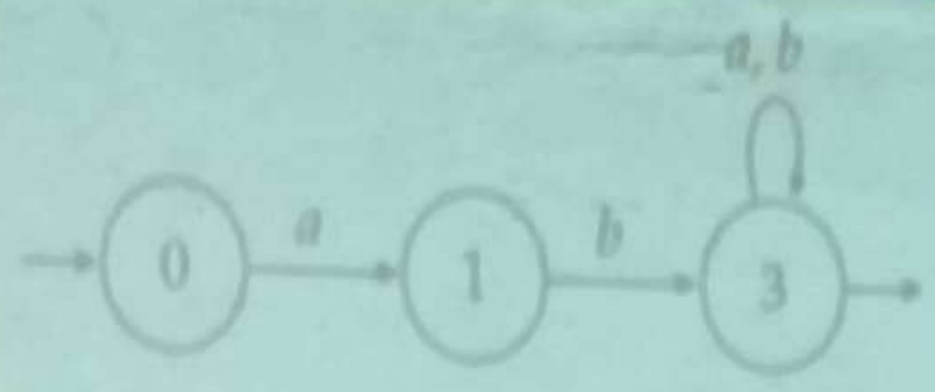
Procédure

- $Q' = Q \cup \{q_p\}$
- $q'_0 = q_0$
- $F' = F$
- $\forall q \in Q, a \in \Sigma, \delta'(q, a) = \delta(q, a)$ si $\delta(q, a)$ existe ; $\delta'(q, a) = q_p$ sinon.
- $\forall a \in \Sigma, \delta'(q_p, a) = q_p$

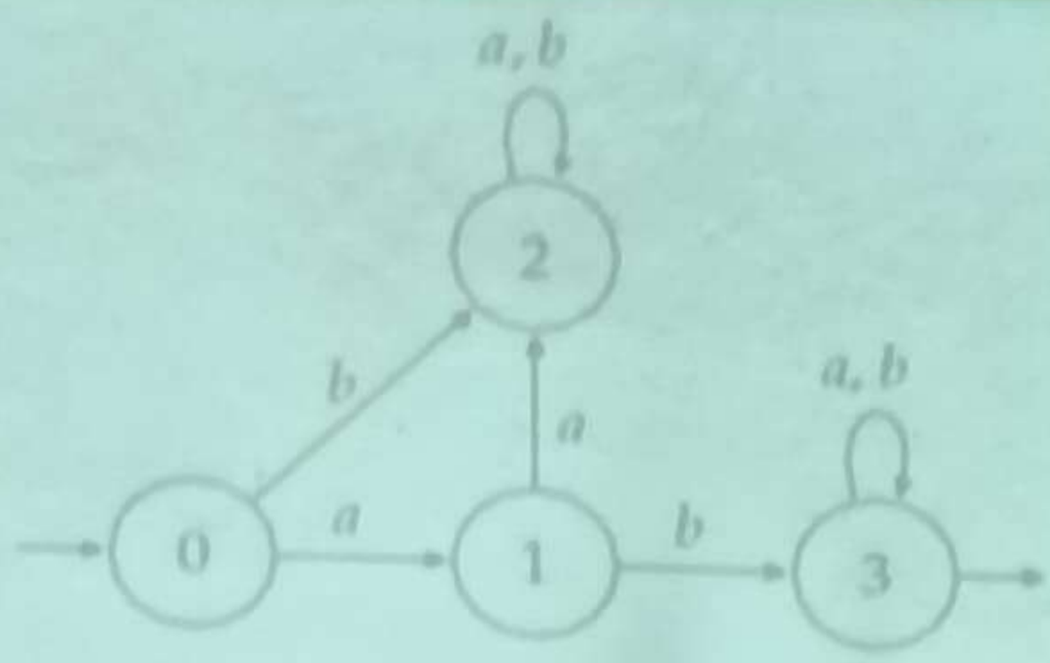
DBeaver CE

DataGrip

Un AFD partiellement spécifié



Un AFDC



DBeaver CE

DataGrip

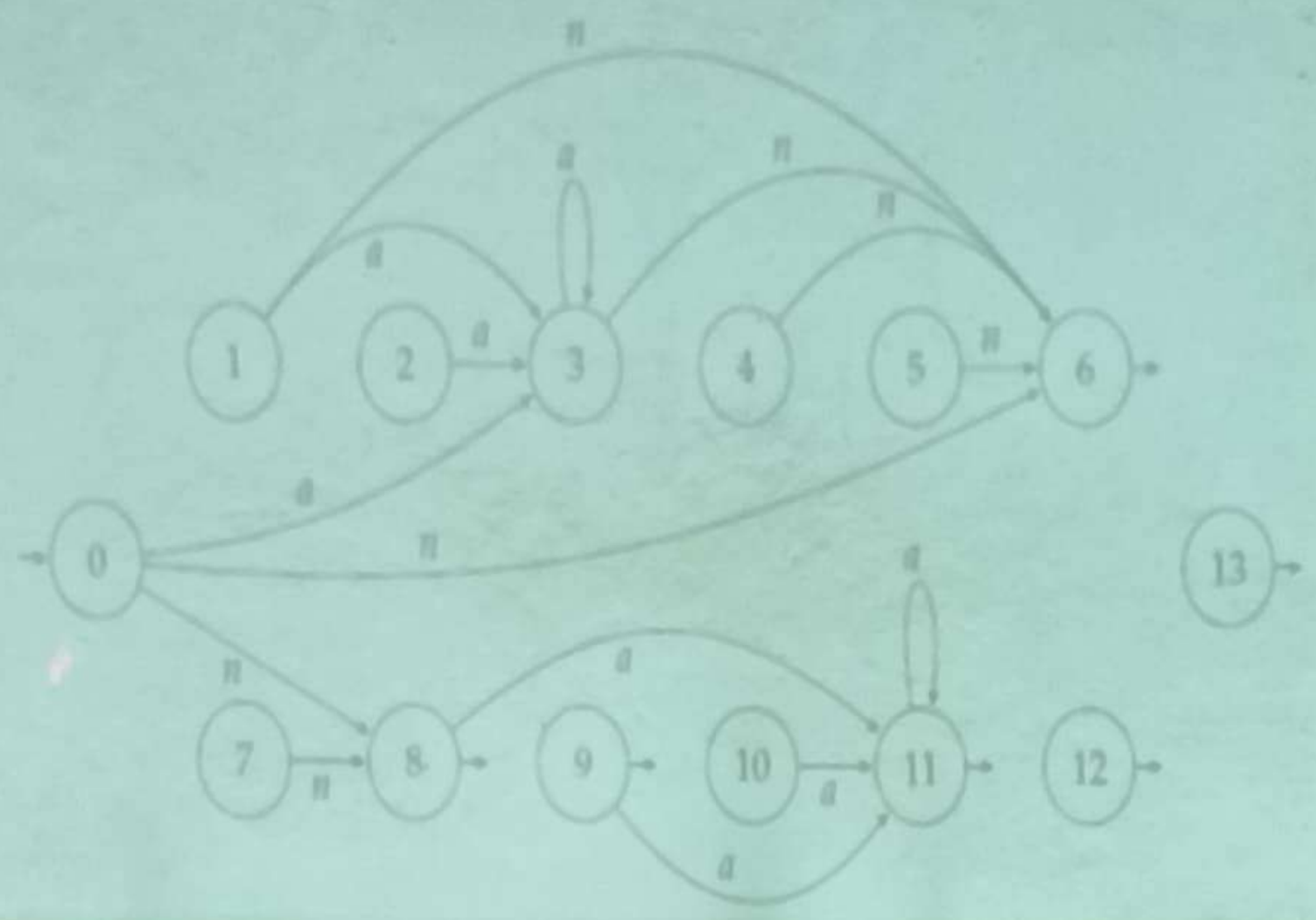
Etats utiles

- Un état q de A est dit **accessible** s'il existe u dans Σ^* tel que $\delta^*(q_0, u) = q$. q_0 est trivialement accessible (par $u = \epsilon$). Un automate accessible est un automate dont tous les états sont accessibles.
- Un état q de A est dit **co-accessible** s'il existe u dans Σ^* tel que $\delta^*(q, u) \in F$. Tout état final est trivialement co-accessible (par $u = \epsilon$). Un automate co-accessible est un automate dont tous les états sont co-accessibles.
- Un état q de A est dit **utile** s'il est à la fois accessible et co-accessible.
- **Un automate émondé** (en anglais **trim automaton**) est un automate dont tous les états sont utiles.

DBeaver CE
DataGrip

Exemple

Etat accessible, co-accessible, utile... Automate émondé



$u_1 \cdots u_n v$, $(\delta(q_1, u_1), u_2 \cdots u_n v) \cdots (\delta(q_n, u_n), v)$

u

$\in \Sigma^*$

$(q_1, uv) \vdash^* (p, v)$

$(q, av) \vdash (\delta(q, a), v)$

DBeaver CE

DataGrip

qu_1

v

$(q, u_1 u_2 u_3 \cdots u_n) \vdash$

$, a, e_{i+1}$

$, b, e_{i+2}$

$\vdash^* (p, \varepsilon)$

$q \xrightarrow{u_1} (\delta(q, u_1))$

Automate fini Non déterministe (AFN / AFND)

De façon informelle, un AFN... Mais pourquoi ?

- Construire un AFD est assez aisé.
- Les combinaisons simples d'automates qui viennent à l'esprit ont un inconvénient majeur : l'objet obtenu n'est pas, dans le cas général, un automate déterministe à nombre fini d'états.
- Pour contourner cette difficulté on étend la notion d'automate en définissant ce qu'on appelle un automate non déterministe (AFN / AFND).

$(\delta(q_1,u_1),\ u_2 \ldots u_n v) \ldots (\delta(q_n,u_n),\ \sigma)$

$(q_1, uv) \vdash^* (q_1, \sigma)$

$(q, av) \vdash (\delta(q,a), v)$

$q u_1$

σ

$(q_1,\ u_1 u_2 u_3 \ldots u_n) \vdash$

$q \xrightarrow{u_1} \delta(q,u_1)$

DBeaver CE

DataGrip

Automate fini Non déterministe (AFN / AFND)

Définition formelle d'un automate fini non déterministe

Un automate fini non-déterministe est défini par un quintuplet $A = (\Sigma, Q, I, F, \delta)$ où :

- Σ est un ensemble fini de symboles (l'alphabet)
- Q est un ensemble fini d'états
- $I \subset Q$ est l'état initial
- $F \subset Q$ sont les états finaux
- $\delta \subset Q \times \Sigma \times Q$ est une relation, chaque triplet $(q, a, q') \in \delta$ est une transition.

Langage reconnu par un automate fini non déterministe

Le langage reconnu par un AFND est défini par :

$$L(A) = \{u \in \Sigma^* \mid \exists q_0 \in I : \delta^*(q_0, u) \cap F \neq \emptyset\}$$

DBeaver CE

DataGrip