Théorie de compilation Analyse lexicale

Dr Abdelmoghit Souissi

Spécification des unités lexicales

3.1- Chaînes et langages :

Définitions générales:

- Un alphabet Σ ou une classe de caractères définit un ensemble fini de symboles.

Exemples : {0,1} : l'alphabet binaire

ASCII : l'alphabet informatique- Une chaîne ou un mot sur un alphabet Σ est une séquence finie de symboles extraits de cet ensemble.

Spécification des unités lexicales

- Longueur d'une chaîne s est notée: $|s|$
- La longueur de la chaîne vide notée ε : $|\varepsilon| = 0$
- L'ensemble des mots sur l'alphabet Σ est noté $\Sigma*$
- Un mot $u \in \Sigma*$ est facteur du mot $w \in \Sigma*$ s'il existe $v, v' \in \Sigma*$ du mot $w \in \Sigma*$
  - tels que: $w = v\ u\ v'$.
- Un mot fini u est périodique
- si $u = x^n$ pour $n \geq 2$. Tout mot non périodique est dit primitif

Spécification des unités lexicales

Soit une chaîne s :

- préfixe de s est une chaîne obtenue en supprimant un nombre quelconque (même nul) de symboles à la fin de s.
- suffixe de s est une chaîne obtenue en supprimant un nombre quelconque (même nul) de symboles au début de s.
- sous-chaîne de s est une chaîne obtenue en supprimant un préfixe et un suffixe.
- sous-suite de s est une chaîne obtenue en supprimant un nombre quelconque (même nul) de symboles non nécessairement consécutifs
- Un langage est un ensemble quelconque de chaînes construites sur un alphabet fixé.

Spécification des unités lexicales

Opérations sur les langages

Soit L et M deux langages:

- Union de L et M : $L \cup M = \{ \forall s / s \in L \text{ ou } s \in M\}$
- Concaténation de L et M : $LM = \{ st / s \in L \text{ et } t \in M$
- Fermeture de Kleene de L : $L^* = \bigcup_{i=0}^{\infty} L_i$
- L* dénote un nombre quelconque (même nul) de concaténation de L.
- On note $L^0 = \{\varepsilon\}$
- Fermeture positive de L : $L+ = \bigcup_{i=1}^{\infty} L_i$

Spécification des unités lexicales

Soit L = {A,B,...,Z} U {a,b,...,z}
et C = {0,1,...9}
A partir de L et C, nous pouvons produire d'autres langages.

- L U C : ensemble des lettres et chiffres,
- LC : ensemble des chaînes constituées d'une lettre suivie d'un chiffre,
- L4 : ensemble des chaînes constituées de 4 lettres,
- C+ : ensemble des entiers naturels,
- L(LUC)* : ensemble des chaînes constituées d'une lettre suivie d'une chaîne de lettres et de chiffres ou d'une chaîne vide.

Les expressions régulières

Les **expressions régulières** sont des outils utilisés dans la **théorie de la compilation**, principalement dans la phase d'**analyse lexicale**, pour définir des modèles de chaînes de caractères (ou *patterns*) et reconnaître des séquences spécifiques dans le code source, comme des mots-clés, des identificateurs, des nombres, etc.

Pourquoi utilise-t-on les expressions régulières dans la compilation ?
Lors de l'analyse lexicale, un programme source est divisé en unités de base appelées **tokens** (par exemple : des mots-clés, des opérateurs, des nombres).

Les **expressions régulières** permettent de décrire ces tokens de manière formelle, en fournissant un moyen de définir ce qu'une chaîne doit ressembler pour être acceptée comme un certain type de token.

Règles des expressions régulières
Symboles de base

- **.** : Correspond à n'importe quel caractère, sauf le caractère de nouvelle ligne (\n).Exemple : a.b correspond à "aXb", "a_b", etc.
- **^** : Indique le début d'une ligne ou d'une chaîne. Exemple : ^abc correspond à "abc" uniquement si "abc" est au début de la chaîne.
- **$** : Indique la fin d'une ligne ou d'une chaîne. Exemple : abc$ correspond à "abc" uniquement si "abc" est à la fin de la chaîne.
- **** : Sert à échapper les caractères spéciaux (comme \. pour rechercher un point littéral).Exemple : \. correspond au caractère "point" lui-même.

Règles des expressions régulières Quantificateurs

- Les quantificateurs définissent combien de fois un caractère ou un groupe de caractères doit apparaître :
- * : Correspond à 0 ou plusieurs occurrences du caractère précédent. Exemple : ab*c correspond à "ac", "abc", "abbc", etc.
- + : Correspond à 1 ou plusieurs occurrences du caractère précédent. Exemple : ab+c correspond à "abc", "abbc", mais pas "ac".
- ? : Correspond à 0 ou 1 occurrence du caractère précédent. Exemple : ab?c correspond à "ac" ou "abc".
- {n} : Correspond à exactement n occurrences du caractère précédent. Exemple : a{3} correspond à "aaa".
- {n,m} : Correspond à entre n et m occurrences du caractère précédent. Exemple : a{2,4} correspond à "aa", "aaa", ou "aaaa".
- {n,} : Correspond à au moins n occurrences du caractère précédent. Exemple : a{2,} correspond à "aa", "aaa", "aaaa", etc.

Règles des expressions régulières
Groupes et alternance

- ( ) : Définissent un groupe capturant. Ce groupe est considéré comme une unité. Exemple : (abc)+ correspond à une ou plusieurs occurrences du groupe "abc".
- | : Opérateur d'alternance qui signifie "ou". Exemple : a|b correspond à "a" ou "b".
- (?:...) : Groupe non capturant, qui regroupe des caractères sans les capturer pour une utilisation ultérieure. Exemple : (?:abc)+ correspond à une ou plusieurs occurrences de "abc", mais ne capture pas ce groupe.

Règles des expressions régulières
Classes de caractères

- [ ] : Définissent une classe de caractères qui correspond à n'importe quel caractère dans la classe. Exemple : [abc] correspond à "a", "b", ou "c".
- [^] : Classe de caractères négative qui correspond à n'importe quel caractère sauf ceux spécifiés. Exemple : [^abc] correspond à tout caractère sauf "a", "b", ou "c".
- [a-z] : Correspond à un intervalle de caractères. Dans cet exemple, de "a" à "z". Exemple : [a-z] correspond à toutes les lettres minuscules.
- \d : Correspond à un chiffre (équivalent à [0-9]).
- \D : Correspond à tout caractère qui n'est pas un chiffre.
- \w : Correspond à un caractère de mot (lettre, chiffre ou underscore). C'est équivalent à [A-Za-z0-9_].
- \W : Correspond à tout caractère qui n'est pas un caractère de mot.
- \s : Correspond à un espace blanc (espace, tabulation, etc.).
- \S : Correspond à tout caractère qui n'est pas un espace blanc.

Règles des expressions régulières Répétition non-gourmande

- Si vous ajoutez un ? après un quantificateur, il devient non-gourmand, c'est-à-dire qu'il capturera le moins de texte possible.
- .*? : Correspond à 0 ou plusieurs occurrences de manière non-gourmande. Exemple : .*? correspond au plus petit nombre de caractères possible. Exemple Pour la chaîne "abc123", l'expression a.*?1 capturera "abc1" (le plus petit nombre de caractères possible entre "a" et "1").
- +? : Correspond à 1 ou plusieurs occurrences de manière non-gourmande. Exemple : Pour la chaîne "abcd123", l'expression a.+?d capturera "abcd".
- ?? : Correspond à 0 ou 1 occurrence de manière non-gourmande.

Comparaison gourmande vs non-gourmande :Gourmande :

- ".*" capturera autant de caractères que possible. Exemple : "a12345b" avec ".*b" capturera "a12345b"

- Non-gourmande : ".*?" s'arrête dès que possible. Exemple : "a12345b" avec ".*?b" capturera "a123b".

Règles des expressions régulières
Ancrages et assertions

- \b : Représente une limite de mot, c'est-à-dire la position entre un caractère de mot (\w) et un caractère non-mot (\W).Exemple : \bword\b correspond à "word" uniquement si c'est un mot complet.
- \B : Représente une absence de limite de mot. Exemple : \Bword\ B ne correspond à "word" que s'il est entouré d'autres caractères de mot.
- (?=...) : Lookahead positif, qui vérifie qu'une certaine condition est satisfaite à venir sans consommer de caractères.
- (?!...) : Lookahead négatif, qui vérifie qu'une certaine condition n'est pas satisfaite à venir.
- (?<=...) : Lookbehind positif, qui vérifie qu'une certaine condition est satisfaite avant sans consommer de caractères.
- (?<!...) : Lookbehind négatif, qui vérifie qu'une certaine condition n'est pas satisfaite avant.

Exemple encrage

^ : Début de la chaîne Cet ancrage correspond au début de la chaîne ou d'une ligne.
Exemple :Expression : ^Bonjour
Chaîne : "Bonjour tout le monde«
Correspond : Oui, car la chaîne commence par "Bonjour".
Chaîne : "Salut Bonjour«
Correspond : Non, car "Bonjour" n'est pas au début.
$: Fin de la chaîne Cet ancrage correspond à la fin de la chaîne ou d'une ligne.
Exemple :Expression : monde$
Chaîne : "Bonjour tout le monde«
Correspond : Oui, car la chaîne se termine par "monde".
La chaine "Bonjour monde tout«
Correspond : Non, car "monde" n'est pas à la fin..

\b : Limite de mot (Word boundary) Cet ancrage correspond à une limite de mot, c'est-à-dire une position entre un caractère alphanumérique et un caractère non-alphanumérique ou une limite de chaîne.
Exemple :Expression : "le chat est ici" \bchat\b :
Correspond : Oui, car "chat" est entouré de limites de mots.
Chaîne : "le chatchat est ici"
Correspond : Non, car "chat " n'est pas isolé.
\B : Pas de limite de mot (Non-word boundary)Cet ancrage correspond à un endroit où il n'y a pas de limite de mot.
Exemple :Expression : \Bchat\B Chaîne : "le chatchat est ici«
Correspond : Oui, car "chat" fait partie d'un mot plus grand.
Chaîne : "le chat est ici«
Correspond : Non, car "chat" est un mot isolé.

Assertions lookahead

Les assertions permettent de vérifier si un certain motif apparaît avant ou après un autre motif sans inclure ce motif dans le résultat capturé.
Positive lookahead : (?=...)Vérifie que ce qui suit correspond à un certain motif sans le consommer.

- Exemple :Expression : \d(?= kg) Chaîne : "5 kg«
- Correspond : Oui, car il y a un chiffre suivi de " kg".
- Chaîne : "5 litres«
- Correspond : Non, car " litres" ne correspond pas à " kg".

Negative lookahead : (?!...)Vérifie que ce qui suit ne correspond pas à un certain motif.

- Exemple :Expression : \d(?! kg) Chaîne : "5 litres«
- Correspond : Oui, car il y a un chiffre non suivi de " kg".
- Chaîne : "5 kg«
- Correspond : Non, car le chiffre est suivi de " kg".

Assertions lookahead

Les assertions permettent de vérifier si un certain motif apparaît avant ou après un autre motif sans inclure ce motif dans le résultat capturé.
Positive lookahead : (?=...)Vérifie que ce qui suit correspond à un certain motif sans le consommer.

- Exemple :Expression : \d(?= kg) Chaîne : "5 kg«
- Correspond : Oui, car il y a un chiffre suivi de " kg".
- Chaîne : "5 litres«
- Correspond : Non, car " litres" ne correspond pas à " kg".

Negative lookahead : (?!...)Vérifie que ce qui suit ne correspond pas à un certain motif.

- Exemple :Expression : \d(?! kg) Chaîne : "5 litres«
- Correspond : Oui, car il y a un chiffre non suivi de " kg".
- Chaîne : "5 kg«
- Correspond : Non, car le chiffre est suivi de " kg".

Assertions lookbehind

Positive lookbehind : (?<=...) Vérifie que ce qui précède correspond à un certain motif.

- Exemple :Expression : (?<=€)\d+ Chaîne : "€50"
- Correspond : Oui, car il y a un nombre précédé de "€".
- Chaîne : "USD50 "
- Correspond : Non, car "USD" ne correspond pas à "€".d.

Negative lookbehind : (?<!...) Vérifie que ce qui précède ne correspond pas à un certain motif.

- Exemple :Expression : (?<!USD)\d+ Chaîne : "50"
- Correspond : Oui, car le nombre n'est pas précédé de "USD".
- Chaîne : "USD50"
- Correspond : Non, car le nombre est précédé de "USD".

Concepts clés des expressions régulières

Les expressions régulières utilisent des symboles spéciaux pour décrire des modèles de texte. Voici les concepts de base :

- Caractères littéraux : Représentent des caractères individuels. Par exemple, l'expression régulière *if* reconnaît la chaîne de caractères *"if"* dans un programme.
- Concatenation : La juxtaposition de deux expressions régulières signifie qu'elles doivent se suivre dans l'ordre. Par exemple, *ab* correspond à la chaîne *"ab".*
- Union (ou) : Le symbole **|** signifie **"ou",** permettant de choisir entre plusieurs options. Par exemple, a|b correspond soit à *"a"* soit à "b".
- Kleene star (*) : Indique que l'expression qui précède peut apparaître zéro ou plusieurs fois. Par exemple, a* correspond à une séquence de zéro ou plusieurs "a", comme *"", "a", "aa", "aaa", etc.*

Concepts clés des expressions régulières

- Plus (+) : Indique que l'expression qui précède doit apparaître une ou plusieurs fois. Par exemple, ***a+*** correspond à ***"a", "aa", "aaa"***, etc., mais pas à "".
- Point d'interrogation (?) : Indique que l'expression qui précède peut apparaître zéro ou une fois. Par exemple, ***a?*** correspond à ***""*** ou ***"a".***
- Parenthèses : Utilisées pour regrouper des sous-expressions. Par exemple, *(ab)** correspond à zéro ou plusieurs occurrences de "ab", "abab", "abababab" .
- Intervalle de caractères : Utilisé pour définir une plage de caractères. Par exemple, [a-z] correspond à n'importe quelle lettre minuscule entre "a" et "z".

Exemples dans un compilateur

Identificateur : Un identificateur (nom de variable ou de fonction) commence par une lettre ou un souligné (_), suivi de lettres, chiffres ou soulignés. On peut décrire cela avec une expression régulière :

- [a-zA-Z_][a-zA-Z0-9_]*

Cela reconnaît des identificateurs comme varName, _myVar, ou x1.

Nombres entiers : Un entier est une séquence de chiffres. L'expression régulière pour les nombres entiers est :

- [0-9]+

Cela correspond à des valeurs comme 123, 0, ou 45678.

Mots-clés : Les mots-clés d'un langage de programmation comme (if, else, ou while) peuvent être définis directement avec des expressions régulières. Par exemple :if|else|while

Cela reconnaît les mots-clés exacts

- if, else, ou while.

Exemples d'expressions régulières:

Un entier signé, Un nombre décimal, un réel

- Un entier signé ou non: (+|-)? (chiffre)+
- Un nombre décimal:(+|-)? (chiffre)+ (.(chiffre)+)?
- Un réel: (+|-)?(chiffre)+(.(chiffre)+)?((e|E)(+|-)?(chiffre)+)? = [+-]?[0-9]+(.[0-9]+)?((e|E)(+|-)?[0-9]+)?

Règles des expressions régulières Exemples concrets d'expressions régulières

- Email : ^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$ Vérifie que l'email a une structure valide avec des lettres, des chiffres, des points, et des tirets, suivis de "@", puis d'un domaine.
- Numéro de téléphone (format simple) : ^\d{3}-\d{3}-\d{4}$Correspond à un format de numéro de téléphone comme "123-456-7890".
- Mot de passe avec au moins une lettre majuscule, une minuscule, un chiffre, et un caractère spécial :
- ^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$ Ce regex vérifie un mot de passe d'au moins 8 caractères contenant une majuscule, une minuscule, un chiffre, et un symbole spécial.

Options de correspondance (modificateurs)

Certains modificateurs changent la façon dont une expression régulière fonctionne :

- i : Rend la correspondance insensible à la casse.
- g : Active la correspondance globale, c'est-à-dire qu'il trouvera toutes les correspondances dans une chaîne.
- m : Active le mode "multi-lignes", où les symboles ^ et $ correspondent respectivement au début et à la fin de chaque ligne, et non pas uniquement au début ou à la fin de toute la chaîne.

Comment sont utilisées les expressions régulières dans un compilateur ?

- Définition des tokens : Chaque token dans le langage est défini par une expression régulière. Par exemple, les identificateurs, les nombres et les opérateurs peuvent tous avoir des expressions régulières associées.
- Analyse lexicale : L'analyseur lexical (ou lexeur) lit le code source caractère par caractère et utilise les expressions régulières pour reconnaître et séparer ces tokens.
- Génération automatique : Des outils comme Lex ou Flex peuvent générer des analyseurs lexicaux automatiques à partir des spécifications d'expressions régulières.

Exemple concret d'utilisation

Supposons que vous ayez un petit extrait de code source comme ceci :
int x = 42;

Voici quelques exemples de tokens et les expressions régulières correspondantes :

- Mot-clé int : Correspond simplement à l'expression régulière int.
- Identificateur x : Utiliser l'expression régulière [a-zA-Z_][a-zA-Z0-9_]* pour reconnaître x.
- Nombre entier 42 : Reconnaître 42 avec l'expression régulière [0-9]+.
- Opérateur = : Correspond à =.
- Opérateur arithmétique + : Correspond à +.

L'analyseur lexical utilise ces expressions régulières pour séparer chaque token et les envoyer à la phase suivante de l'analyse (l'analyse syntaxique).

En résumé, les expressions régulières sont des outils puissants utilisés dans les compilateurs pour décrire les modèles de chaînes de caractères qui composent les différents éléments du code source. Elles sont essentielles pour définir et reconnaître les tokens dans la phase d'analyse lexicale.

Les automates déterministes (DFA) et non déterministes (NFA)

sont deux types d'automates finis utilisés pour reconnaître des langages réguliers dans la théorie des langages et des automates.

1. Automate Déterministe (DFA - Deterministic Finite Automaton) Un DFA est un automate fini dans lequel, pour chaque état et chaque symbole de l'alphabet, il existe exactement une transition vers un autre état ou vers le même état.
2. Cela signifie qu'à chaque étape de traitement d'une chaîne, il n'y a qu'une seule option à suivre, ce qui rend l'automate déterministe.
3. Structure:Un DFA est défini comme un quintuplet (Q,Σ,δ,q_0,F)où :Q : un ensemble fini d'états. Σ : un alphabet fini de symboles. δ : une fonction de transition $\delta:Q\times\Sigma\rightarrow Q$, qui donne un état pour chaque couple (état, symbole).
4. q_0 : l'état initial, $q_0\in Q$
5. F : un ensemble d'états finaux (ou acceptants), $F\subseteq Q$

Les automates déterministes (DFA)

Exemples de DFA :Prenons un exemple simple d'un DFA qui reconnaît les chaînes composées uniquement de "a" et "b" et qui finissent par un "b".
DFA pour reconnaître les chaînes qui se terminent par un "b" :
Alphabet : Σ={a,b} États : Q={q_0,q_1} État initial : q_0 État final : q_1 .
Fonction de transition δ

$\delta(q_0,a)$=q0
$\delta(q_0,b)$=q1
$\delta(q_1,a)$=q0
$\delta(q_1,b)$=q1

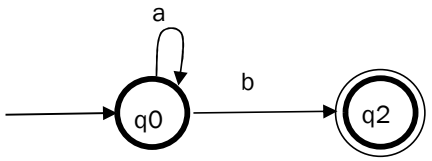


Illustration :Si la chaîne est "ab", elle est acceptée. Si la chaîne est "aa", elle n'est pas acceptée car elle ne se termine pas par un "b".

Fonctionnement :Le DFA commence dans l'état q_0 . Si la chaîne se termine par un "b", il atteint l'état q_1 , qui est un état acceptant. Si la chaîne ne se termine pas par un "b", le DFA finit dans q_0 , qui n'est pas un état acceptant.

Les automates déterministes (NFA)

- Un NFA est un automate fini dans lequel, pour un état donné et un symbole donné, il peut y avoir plusieurs transitions possibles ou même aucune.

- De plus, un NFA peut avoir des transitions "epsilon" (ϵ), c'est-à-dire des transitions qui se produisent sans lire de symbole. Contrairement au DFA, où il n'y a qu'une seule option à suivre à chaque étape, un NFA peut suivre plusieurs chemins possibles simultanément.

- Structure d'un NFA :Un NFA est défini de manière similaire à un DFA, mais avec une différence importante dans la fonction de transition :δ : une fonction de transition $\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$, qui donne un ensemble d'états pour chaque couple (état, symbole). Un NFA peut aller vers plusieurs états simultanément, ou rester dans le même état avec une transition ϵ.

Exemple NFA

Prenons un exemple d'un NFA qui reconnaît les chaînes qui contiennent au moins un "b". NFA pour reconnaître les chaînes qui contiennent au moins un "b" :Alphabet : Σ={a,b} États : Q={q_0,q_1} | État initial : q_0 État final : q_1
Fonction de transition:

$\delta(q_0,a)=\{q_0\}$
$\delta(q_0,b)=\{q_1\}$
$\delta(q_1,a)=\{q_1\}$
$\delta(q_1,b)=\{q_1\}$

Illustration :Si la chaîne est "aaab", elle est acceptée. Si la chaîne est "aaa", elle n'est pas acceptée car elle ne contient pas de "b".

Fonctionnement :Le NFA commence dans l'état q0q 0 . Dès qu'il lit un "b", il passe dans l'état q_1 , qui est un état acceptant. Contrairement au DFA, le NFA peut suivre simultanément plusieurs transitions.

Différences entre DFA et NFA

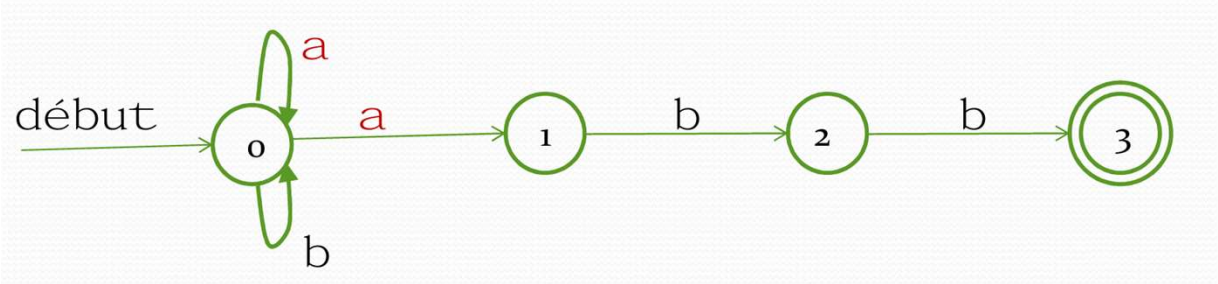
- Dans un DFA, il n'y a qu'un seul chemin possible pour chaque état et symbole. Il peut y avoir plusieurs chemins possibles, ou même des transitions sans lire de symbole (ϵ-transitions).
- Complexité :Les DFA peuvent nécessiter plus d'états pour représenter certains langages par rapport aux NFA, car ils doivent définir un état unique pour chaque combinaison de symboles possibles.
- Équivalence :Tout langage reconnu par un NFA peut également être reconnu par un DFA. En théorie, les NFA et les DFA sont équivalents en termes de puissance d'expression. Cependant, convertir un NFA en DFA peut entraîner une explosion exponentielle du nombre d'états (le pire des cas étant $2n$ états pour un DFA obtenu à partir d'un NFA de n états).

Automates à états finis (AEF)

Automates à états finis non déterministes (AFN)

Exemple

L'AFN qui reconnait le langage défini par l'expression régulière : abb



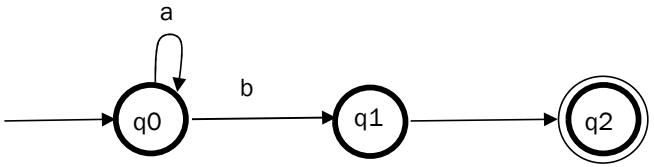
Nous pouvons représenter un AFN par une table de transition, dont les lignes correspondent aux états et les colonnes aux symboles d'entrée et à ε.

Symbole/ Etat	a	b	ε
0	{0,1}	{0}	-
1	-	{2}	-
2	-	{3}	-
3	-	-	-

Exemple AFD

Considérons un exemple d'AFD qui reconnaît les mots composés de zéro ou plusieurs a suivis par un b. L'alphabet est {a, b}.
Définition de l'AFD : États : {q0, q1, q2} ; Alphabet : {a, b} ; État initial : q0 ; État final : q2
Transitions :
De q0, en lisant a, l'automate reste dans q0.
De q0, en lisant b, l'automate passe à l'état q1.
De q1, en lisant un autre symbole (peu importe a ou b), il passe à l'état final q2.

Tableau de transition :



Etat	a	b
q0	q0	q1
q1	q2	q2
q2	--	--

- q0 est l'état initial.
- Si l'automate arrive dans q2, le mot est accepté.

AFD dans la théorie de compilation

Dans un compilateur, les AFD sont souvent utilisés pour reconnaître les lexèmes dans un programme source.
Chaque état peut représenter un stade particulier de la reconnaissance d'un lexème, et en fonction du symbole lu, l'automate passe à un nouvel état jusqu'à ce qu'un lexème complet soit reconnu.

Exemple dans un compilateur :Reconnaître un mot-clé comme if, else, ou un identifiant.
Reconnaître des nombres entiers ou à virgule flottante.

L'AFD parcourt le code caractère par caractère et déplace son état en fonction de ce qu'il lit, jusqu'à ce qu'il ait identifié une unité lexicale (lexème).