

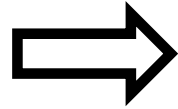
Compilation

Introduction & analyse lexicale

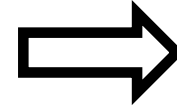
Pourquoi la compilation ?



Le binaire est difficile à
utiliser par les
programmeurs



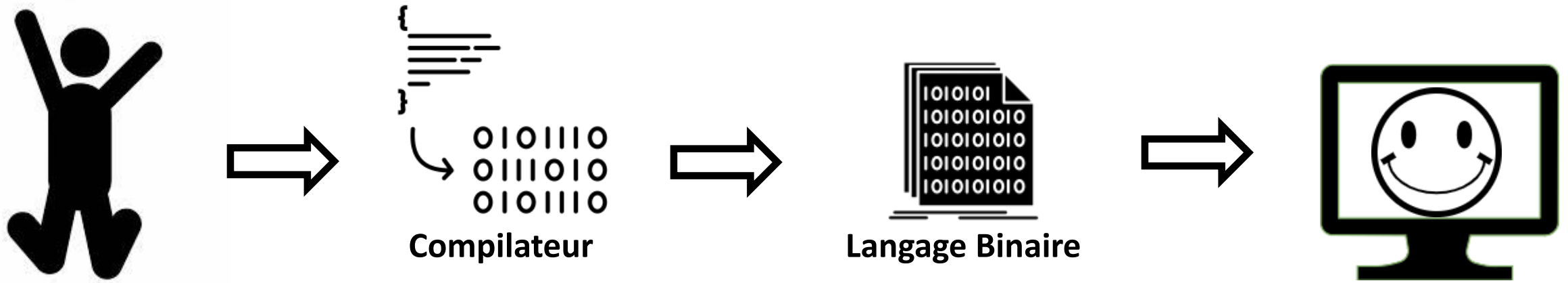
Langage Binaire



L'ordinateur comprends
le binaire
(00011101010)

*Un compilateur permet de faciliter de la programmation
sans utiliser le binaire*

Pourquoi la compilation ?



Un compilateur traduit à partir d'un langage haut niveau vers un langage machine (binaire)

Pourquoi la compilation ?

- Construction des nouveaux langages de programmation
- Le compilateur montre l'application réussie de la théorie à des problèmes pratiques (Théorie de langages)
- Utilisation dans d'autres problèmes (Validation des données, Evaluation des expressions mathématique)
- Comprendre comment aborder les problèmes complexes

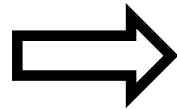
C'est quoi la compilation ?

- Un compilateur traduit **un programme source** écrit dans un langage à **un autre programme cible équivalent** écrit dans un autre langage. tout en **signalent les erreurs détectées** du programme source.

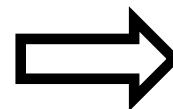


C'est quoi la compilation ?

```
#include <stdio.h>
int main(){
    printf("hello world\n");
    return 0;
}
```

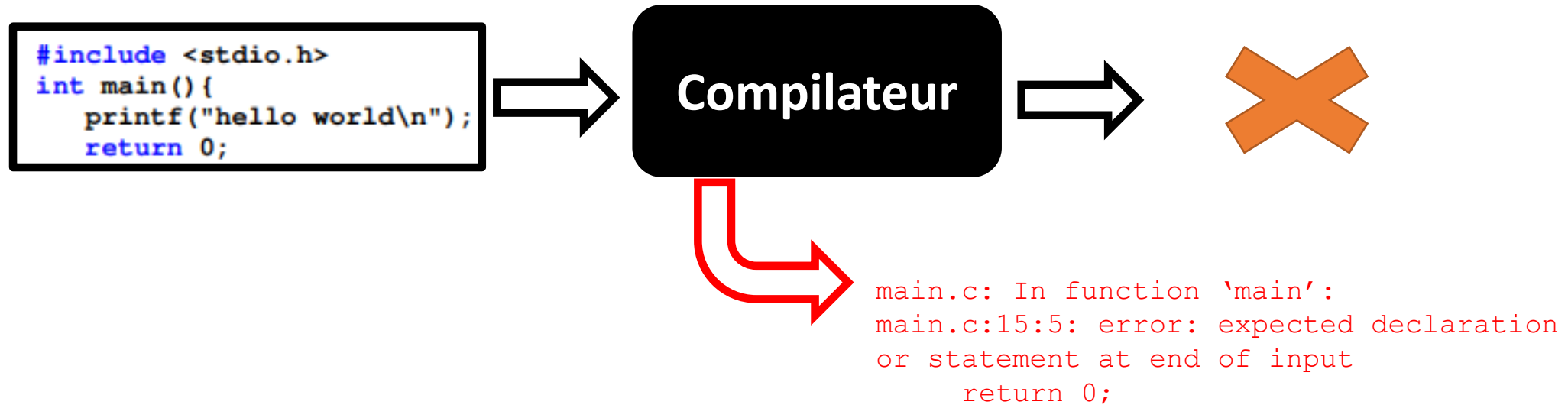


Compilateur

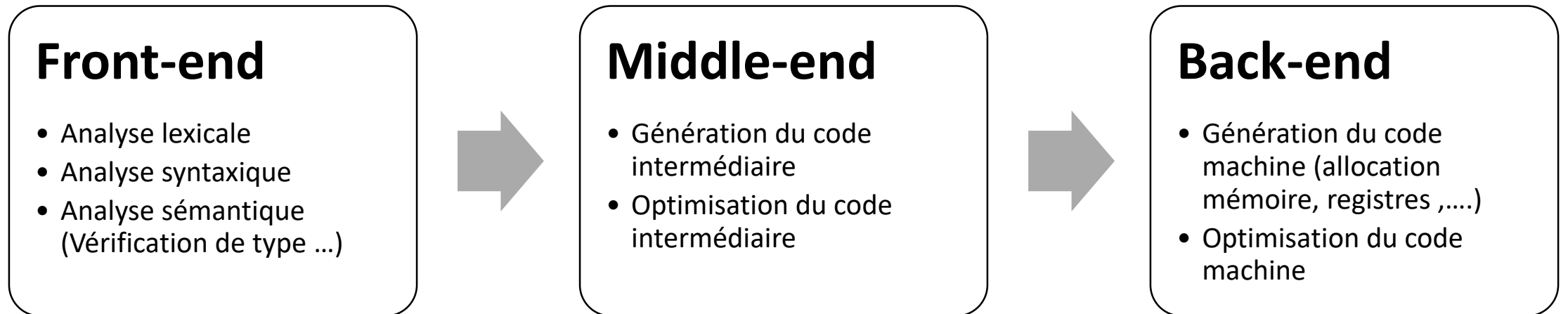


```
// Assembly code
.LC0:
    .string "hello world"
    .text
.globl main
    .type    main, @function
main:
    pushq    %rbp
    movq     %rsp, %rbp
    movl     $.LC0, %edi
    call     printf
    movl     $0, %eax
    leave
    ret
```

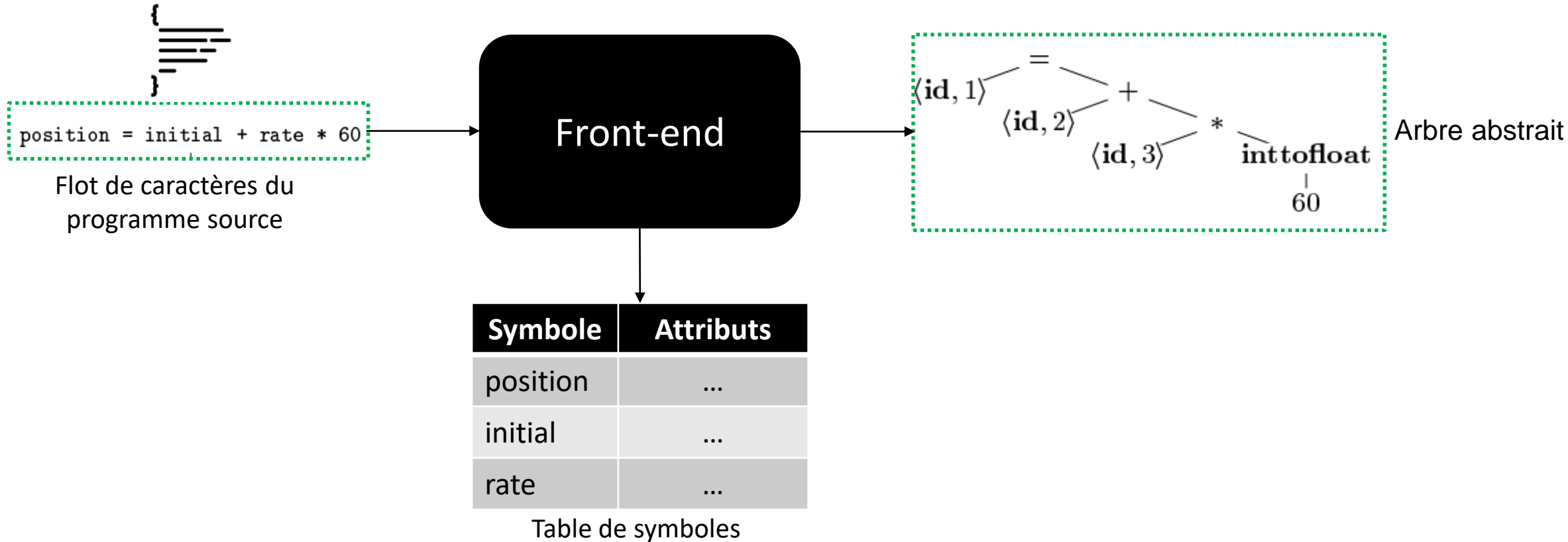
C'est quoi la compilation ?



Structure d'un compilateur

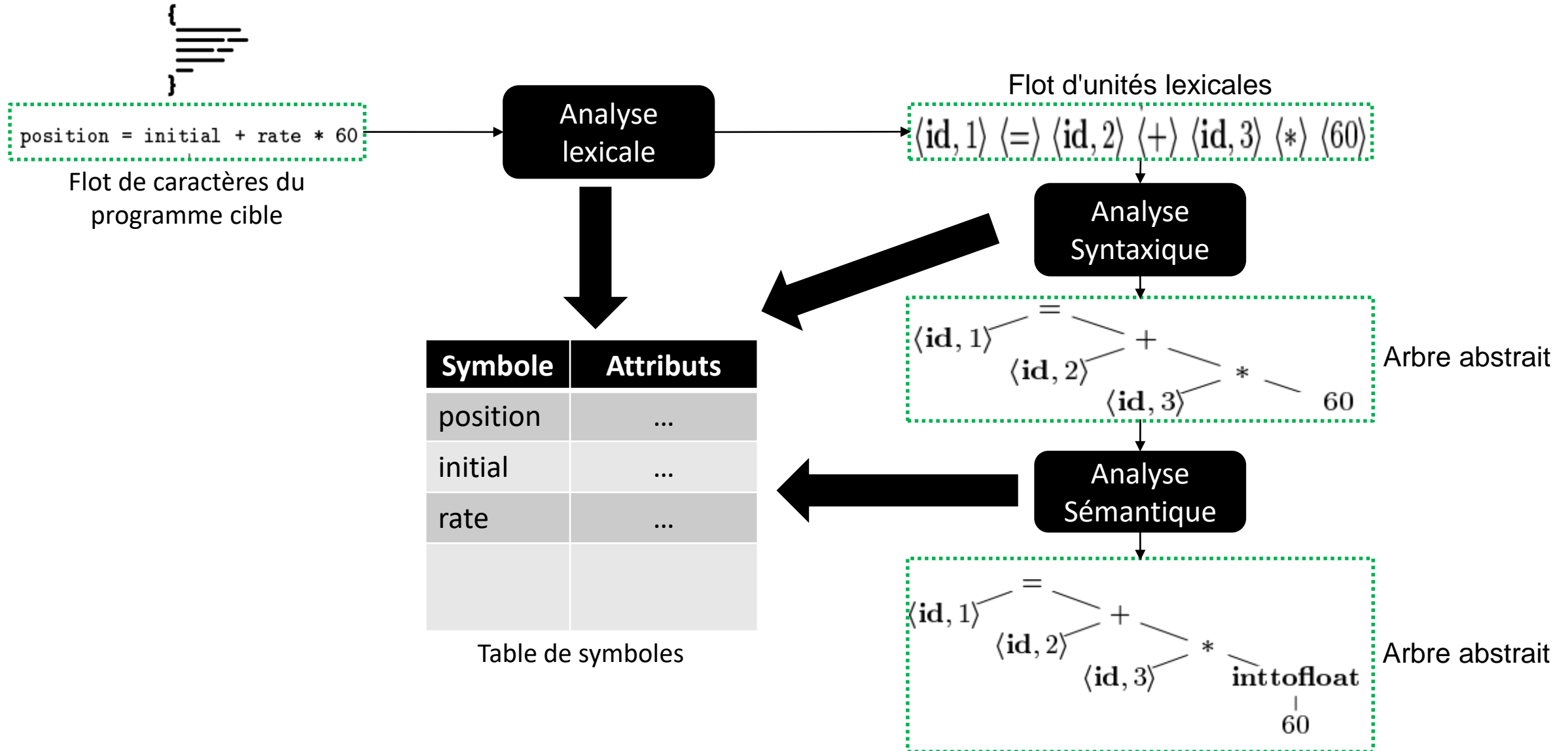


Structure d'un compilateur (Front-end)

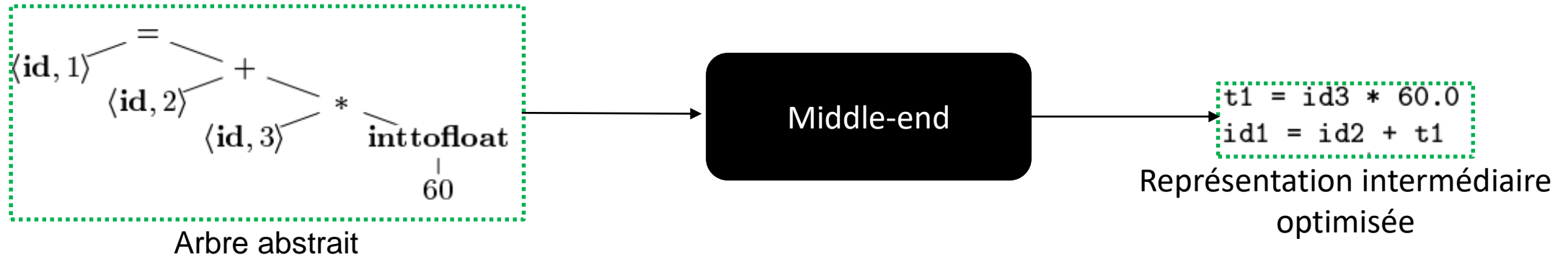


Le front-end essaye de comprendre le programme source

Structure d'un compilateur (Front-end)

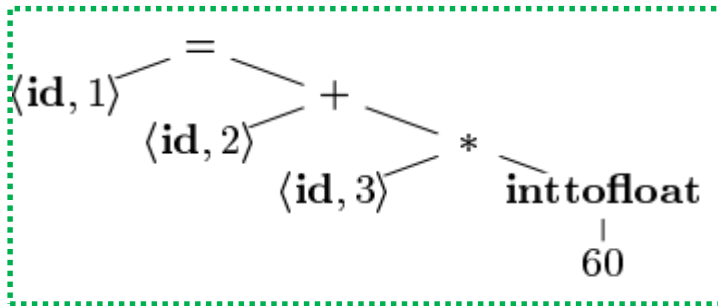


Structure d'un compilateur (Middle-end)



- Représentation intermédiaire avec trois opérandes
- Représentation indépendante de la machine
- Portabilité du programme
- **Exemple: Java Byte Code**

Structure d'un compilateur (Middle-end)



Arbre abstrait

Générateur de code
intermédiaire

Représentation intermédiaire

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

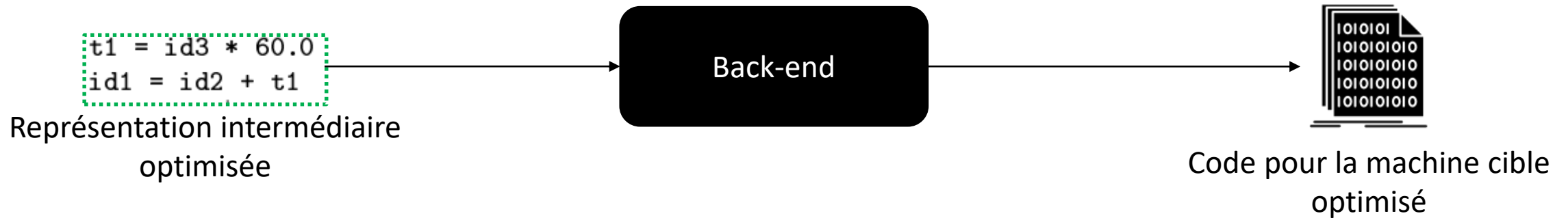
Optimiseur de code
indépendant de la
machine

```
t1 = id3 * 60.0
id1 = id2 + t1
```

Représentation intermédiaire
optimisée

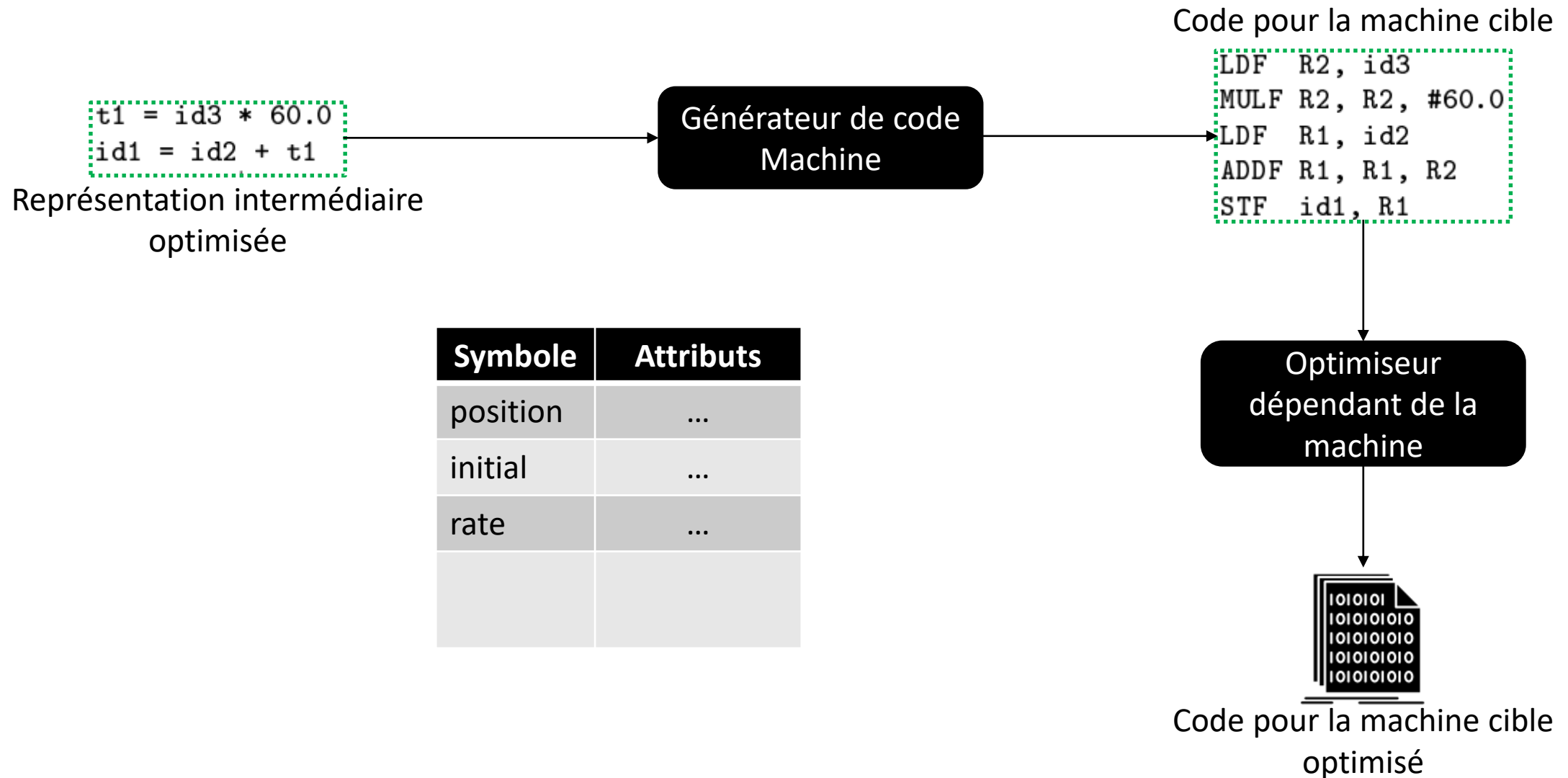
Symbole	Attributs
position	...
initial	...
rate	...

Structure d'un compilateur (Middle-end)



Le back-end essaye de projeter la représentation intermédiaire vers le langage cible

Structure d'un compilateur (Back-end)



Analyse lexicale

C'est quoi l'analyse lexicale ?

- Regrouper les caractères du programme source en lexèmes
- Produire des unités lexicales (Token) du programme source
Unité lexical : < Nom de l'unité lexical, valeurs Attributs >
- Le flot d'unités lexicales est envoyé à l'analyseur syntaxique
Unité lexicale 1, Unité lexicale 2, Unité lexicale N
- L'analyseur lexical interagit aussi avec la table de symboles
Informations à propos les identificateurs (nom, type,)
- Signaler les erreurs lexicales & Supprimer les caractères intitules (espaces, saut de lignes ...)

Exemple

`position = initial + rate * 60`

- Lexème 1 : `'position'` → Unité lexicale 1 **< id, 1 >**
 - **id**: nom abstrait signifiant **identificateur**
 - **1**: attribut qui référence l'entrée de la table de symboles associe à position
- Lexème 2: `'='` → Unité lexicale 2 **< = >**
 - **=** : nom abstrait signifiant **affectation**
 - Absence d'attributs
- Lexème 3 : `'initial'` → Unité lexicale 3 **< id, 2 >**
- Lexème 4: `'+'` → Unité lexicale 4 **< + >**
- Lexème 5: `'rate'` → Unité lexicale 5 **< id, 3 >**
- Lexème 6: `'*'` → Unité lexicale 6 **< * >**
- Lexème 7: `'60'` → Unité lexicale 7 **< nbr, 60 >**

Exemple

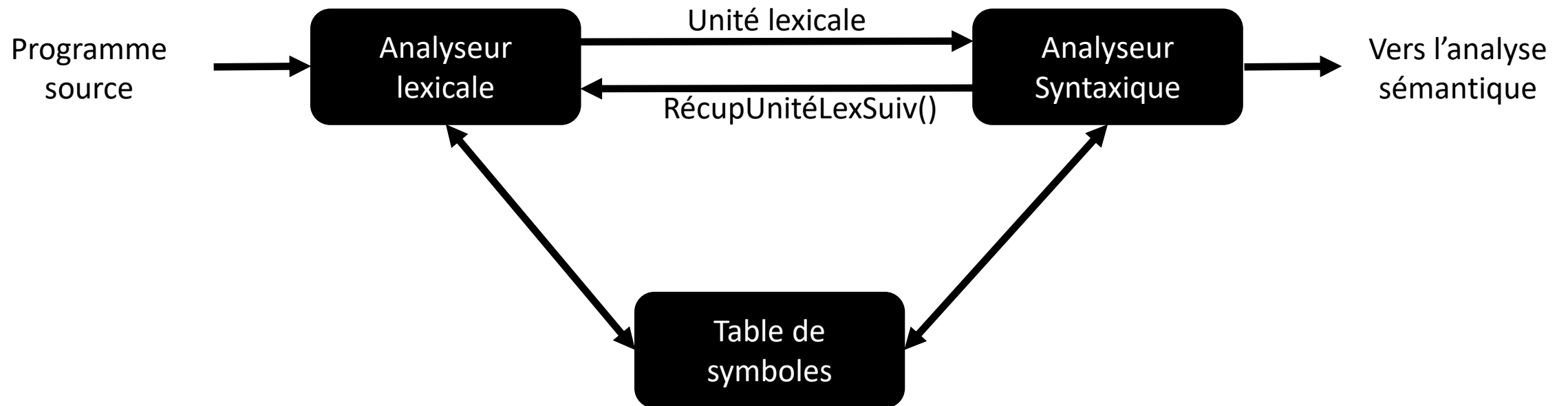
- Chaîne des unités lexicales

< id, 1 >, < = >, < id, 2 >, < + >, < id, 3 >, < * >, < nbr, 60 >

- Table de symbols

Symbole	Attributs
position	...
initial	...
rate	...

C'est quoi l'analyse lexicale ?



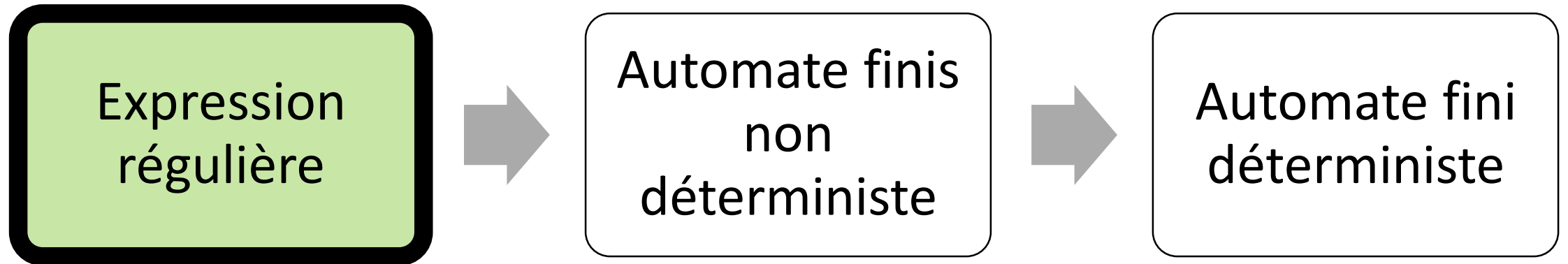
Question

Comment reconnaître les lexèmes dans le programme source ?

Modèles mathématique pour spécifier les unités lexicales

Expressions régulières + Automates

Construction d'un analyseur lexicale



Expressions régulières

- Σ est l'ensemble d'alphabets utilisé
- **Cas de base :**
 - $a = \varepsilon$ est une expression régulière: $L(a) = \{\varepsilon\}$ i.e.: chaîne vide "
 - Si $a \in \Sigma$ alors a est une expression régulière: $L(a) = \{a\}$
- **Induction:**
 - **Alternance :** a et b sont des expressions régulières alors :
 $c = a|b$ est une expression régulière : $L(c) = L(a) \cup L(b)$
 - **Concaténation:** a et b sont des expressions régulières alors :
 $c = ab$ est une expression régulière : $L(c) = L(a)L(b)$
 - **Répétition:** a une expression régulières alors :
 $c = a^*$ est une expression régulière : $L(c) = \underbrace{L(a)L(a) \dots L(a)}_{\text{Répétition 0 } (\varepsilon \in c) \text{ ou plusieurs fois}}$

Exemples

- $a = if$
 - $L(a) = \{ 'if' \}$
- $a = (01)^*$
 - $L(a) = \{ \varepsilon, 01, 0101, 010101, 01010101, \dots, \}$
- **Chiffre** = $0|1|2|3|4|5|6|7|8|9$
 - $L(\text{Chiffre}) = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- **Nbr** = $(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^* = \text{Chiffre Chiffre}^*$
 - $L(\text{Nbr}) = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 00, 01, 02 \dots, 4651328, \dots\}$

Expressions régulières (Extension)

- a^+ est une expression régulière : $L(a^+) = \underbrace{L(a)L(a) \dots L(a)}_{\text{Répétition 1 ou plusieurs fois}}$
- $a? \Leftrightarrow a|\varepsilon$ optionnelle (0 ou 1 occurrence) : $L(a?) = \{\varepsilon, a\}$
- $[ab] \Leftrightarrow a|b$
- $[a-z] \Leftrightarrow a|b|c|d|e|.....|z$ (tous les lettres entre a et z)
- $[a-zA-Z] \Leftrightarrow a|b|c|d|e|.....|z|A|B|C|D|E|.....|Z$

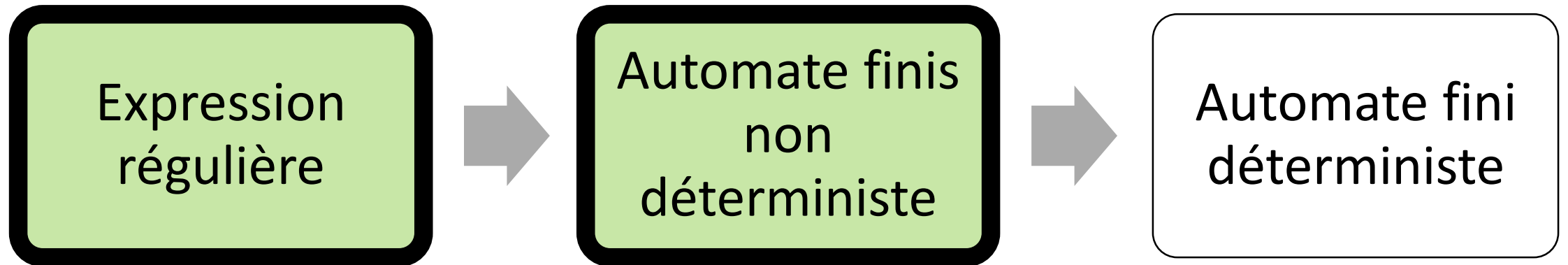
Exemples

- **Id** = $[a-z][a-z0-9]^*$
- **Chiffre** = $[0-9]$
- **NbrEntier** = $[0-9]^+$
- **NbrRéel** = $(+|-|\varepsilon)[0-9]^+ \text{'.'}[0-9]^+$
= $[+|-]?[0-9]^+ \text{'.'}[0-9]^+$

Résumé

Expression régulière	Signification
ε	Chaine vide
$a \in \Sigma$	Alphabet
a b [ab]	Alternance: choix entre a ou b
ab	Concaténation: a suivi par b
a*	Répétition de a 0 ou plusieurs fois
a+	Répétition strictement positive de a 1 ou plusieurs fois
a?	a est optionnelle (0 ou 1 occurrence)
[a-z] [a-zA-Z] [0-9]	Ensemble des lettres minuscules Ensemble des lettres minuscules et majuscules Tous les chiffres

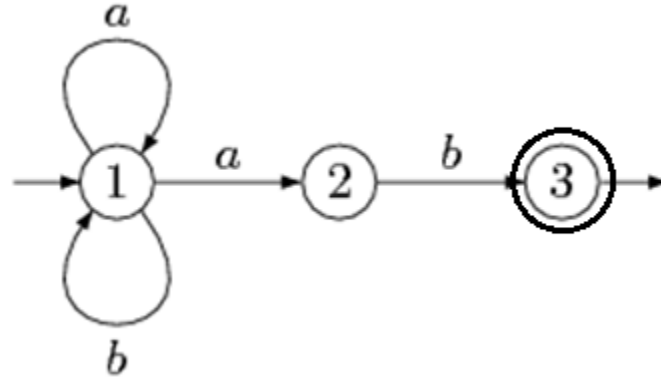
Construction d'un analyseur lexicale



Automate finis non déterministe (AFN)

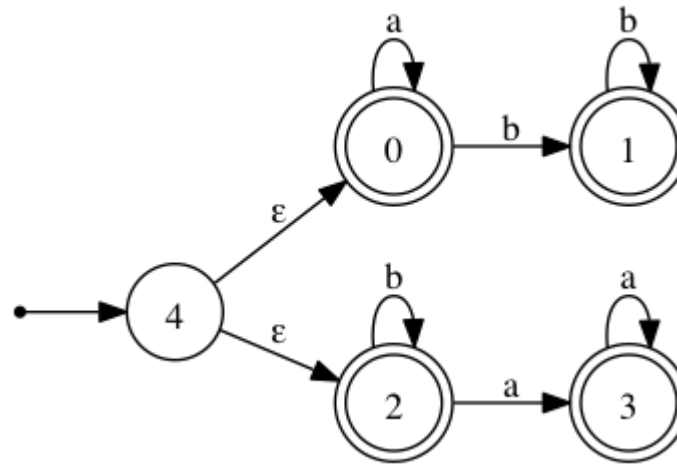
- $AFN : (S, \Sigma, \delta, s_0, F)$
- S : un ensemble fini d'états
- Σ : ensemble d'alphabets
- δ : Fonction de transition pour chaque état $s_i \in S$ et pour chaque alphabet $a \in \Sigma$ ou ε un sous-ensemble d'état $S_i \subset P(S)$:
$$S \times (\Sigma \cup \{\varepsilon\}) \xrightarrow{\delta} P(S)$$
$$\delta(s_i, a) = S_i$$
- $s_0 \in S$: état initiale (état de départ)
- $F \subset S$: ensemble des états finaux (des états d'acceptation)

Exemple



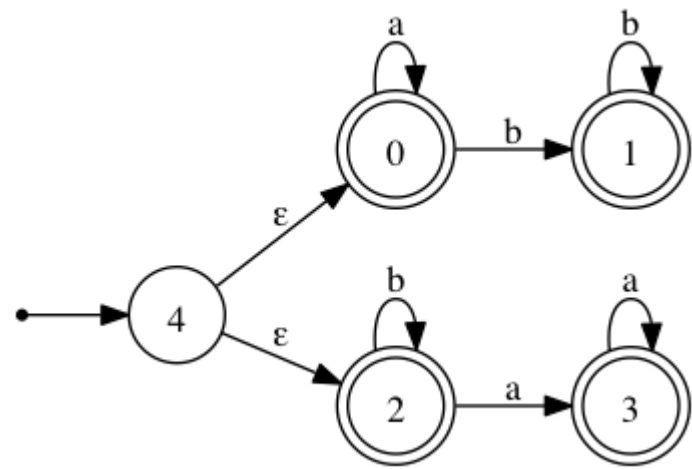
- $AFN : (S, \Sigma, \delta, s_0, F)$
- $S = \{1, 2, 3\}$
- $\Sigma = \{a, b\}$
- δ : Fonction de transition
 - $\delta(1, a) = \{1, 2\}, \delta(1, b) = \{1\}$
 - $\delta(2, b) = \{3\}$
- $s_0 = 1$
- $F = \{3\}$
- $L(AFN) = (a|b)^*ab$

Exemple



- $AFN : (S, \Sigma, \delta, s_0, F)$
- $S = \{0, 1, 2, 3\}$
- $\Sigma = \{a, b\}$
- δ : Fonction de transition
 - $\delta(0, a) = \{0\}, \delta(0, b) = \{1\}$
 - $\delta(1, b) = \{1\}$
 - $\delta(2, a) = \{3\}, \delta(2, b) = \{2\}$
 - $\delta(3, a) = \{3\}$
 - $\delta(4, \varepsilon) = \{0, 2\}$
- $s_0 = 4$
- $F = \{0, 1, 2, 3\}$
- $L(AFN) = \varepsilon|(a^+|a^*b^+)|(b^+|b^*a^+)$

Table de transition

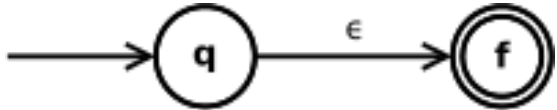
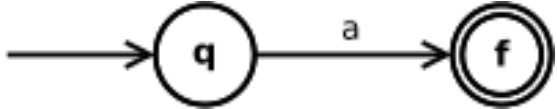


	a	b	ε
0	{0}	{1}	\emptyset
1	\emptyset	{1}	\emptyset
2	{3}	{2}	\emptyset
3	{3}	\emptyset	\emptyset
4	\emptyset	\emptyset	{0,2}

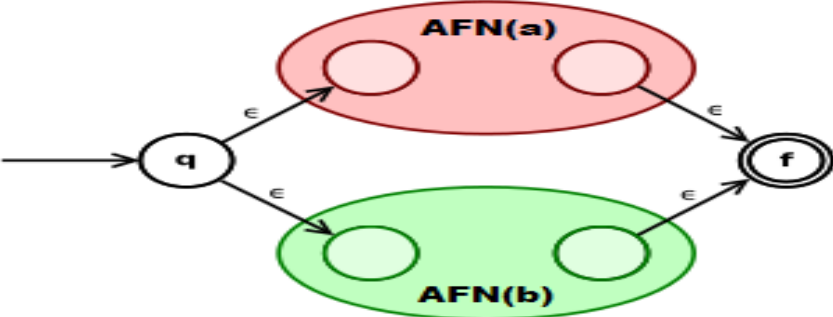
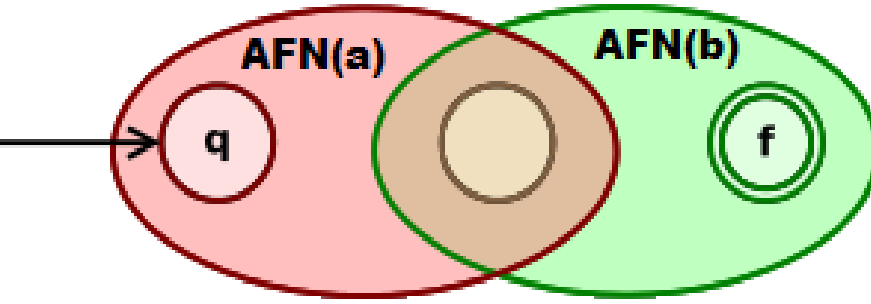
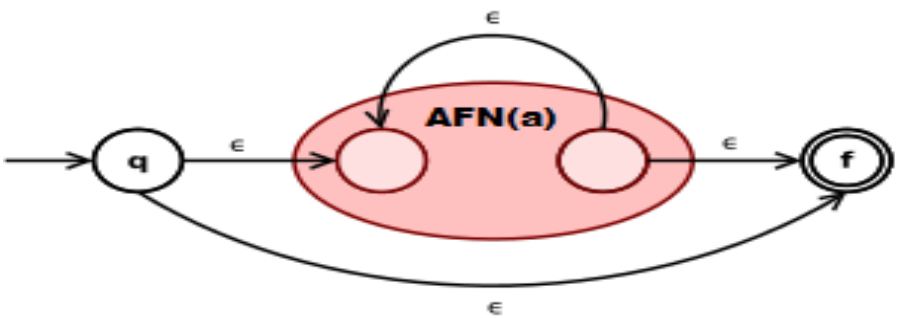
Expression régulière vers AFN

Algorithme de Thompson

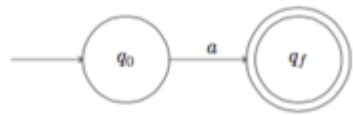
- Cas de base

Expression régulière	AFN
ε	
$a \in \Sigma$	

Expression régulière vers AFN

Expression régulière	AFN
Alternance : $a b$	
Concaténation : ab	
Répétition : a^*	

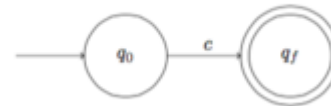
Example: ab/c^*



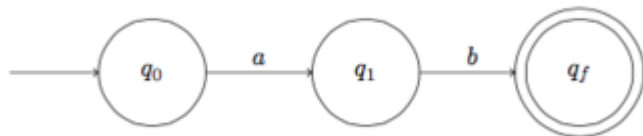
a



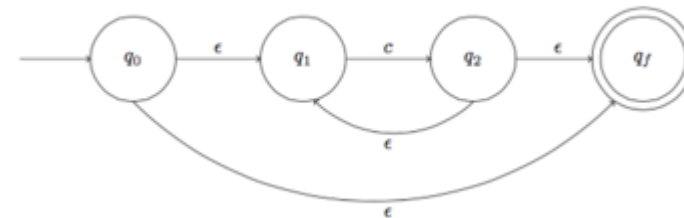
b



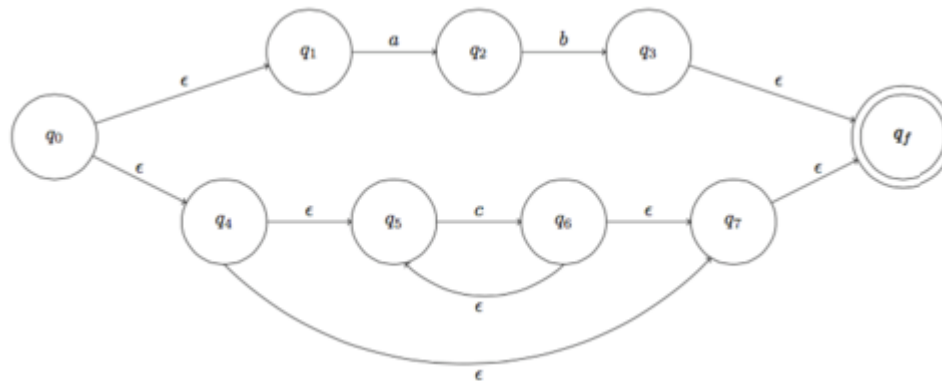
c



ab



c^*



ab/c^*

Avantages et les inconvénients

- Avantages

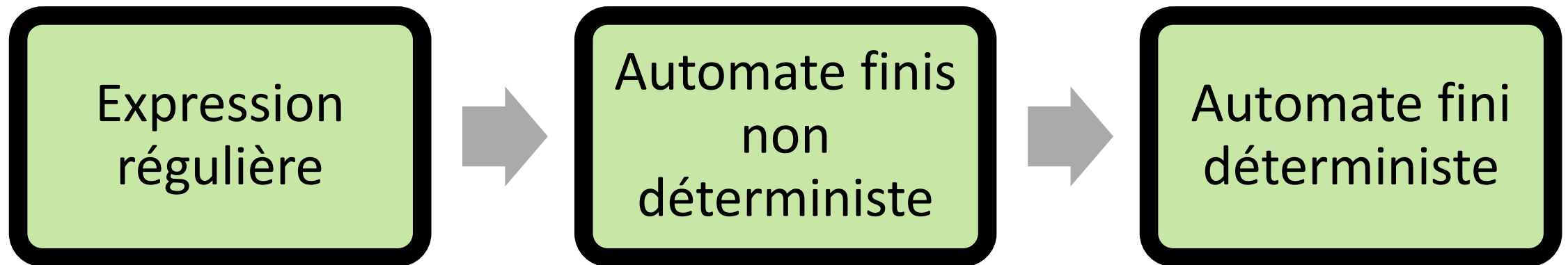
- Passage automatique à partir une expression régulière
- Nombre petit d'états



- Inconvénients

- Plusieurs états peut être active en même temps à cause de ε –transitions
- Plusieurs transitions sont possibles avec un seul alphabet (**non déterministe**)
- Temps de simulation élevé → Temps de reconnaissance élevé

Construction d'un analyseur lexicale



Automate fini déterministe (AFD)

- $AFN : (S, \Sigma, \delta, s_0, F)$
- S : un ensemble fini d'états
- Σ : ensemble d'alphabets
- δ : Fonction de transition pour chaque état $s_i \in S$ et pour chaque alphabet $a \in \Sigma$
un seul état d'état $s_j \in S$:
$$S \times (\Sigma) \xrightarrow{\delta} S$$
$$\delta(s_i, a) = s_j$$
- $s_0 \in S$: état initiale (état de départ)
- $F \subset S$: ensemble des états finaux (des états d'acceptation)

AFN \rightarrow AFD

- ε – *fermeture*(e) : ensemble des états de l'AFN qu'on peut atteindre à partir de e avec ε -transitions
- ε – *fermeture*(T) : ensemble des états de l'AFN qu'on peut atteindre à partir de tous les états de $e \in T$ avec ε -transitions
- *Trans*(T, a) : Ensemble des états vers lesquels il y a une transition avec l'alphabet a à partir de tous les états de $e \in T$

AFN \rightarrow AFD

$$s_0 = \varepsilon - \text{fermeture}(e_0)$$

$$S = \{s_0\}$$

Tant que (**il existe un état non marqué s dans S**) faire

Marquer s

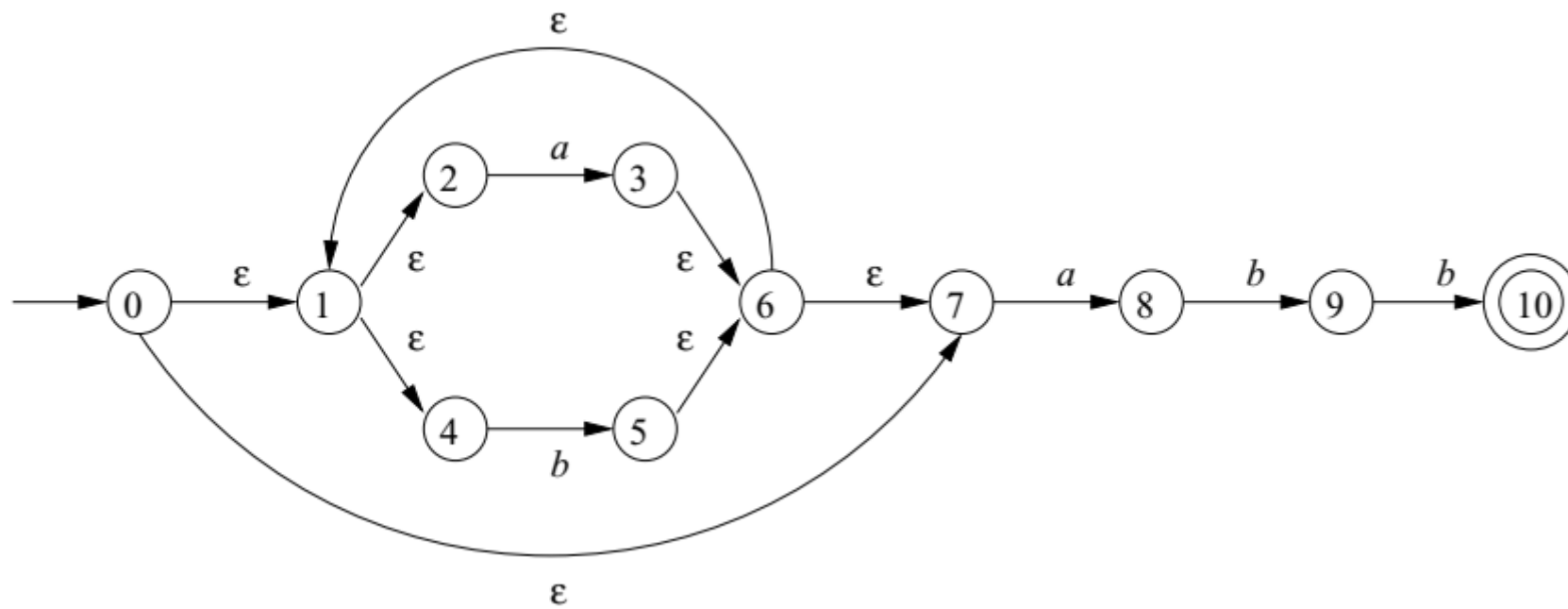
Pour chaque **alphabet $a \in \Sigma$** faire

$$s^* = \varepsilon - \text{fermeture}(\text{Trans}(s, a))$$

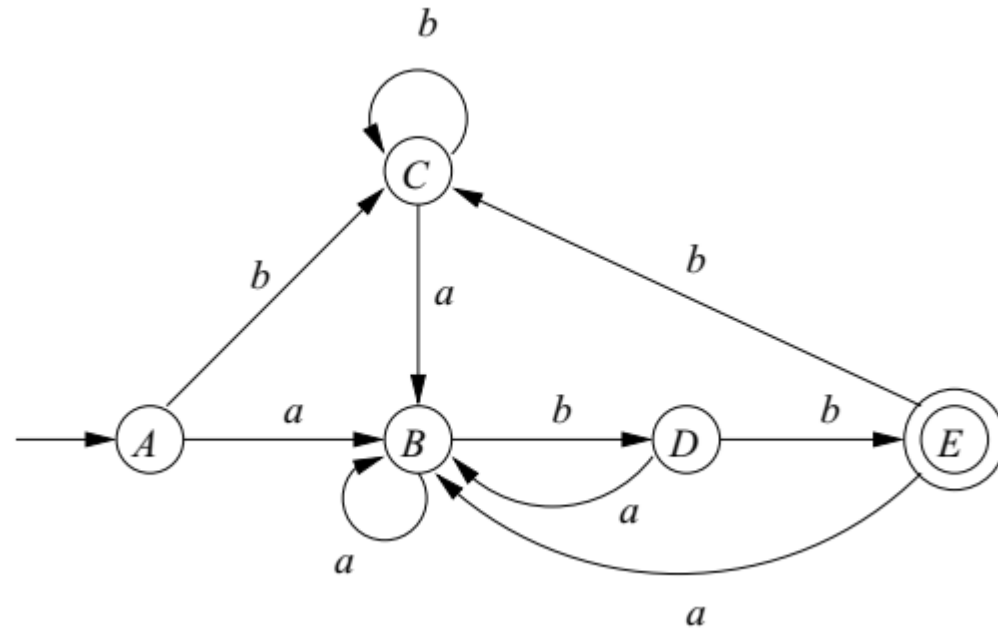
Si (**s^* n'est pas dans S**) Alors /*Nouveau état*/
 $S = S \cup \{s^*\}$

$$\delta(s, a) = s^*$$

Example



Exemple



Etat AFN	Etat AFD	<i>a</i>	<i>b</i>
{0, 1, 2, 4, 7}	A	B	C
{1, 2, 3, 4, 6, 7, 8}	B	B	D
{1, 2, 4, 5, 6, 7}	C	B	C
{1, 2, 4, 5, 6, 7, 9}	D	B	E
{1, 2, 4, 5, 6, 7, 10}	E	B	C

Avantages et les inconvénients

- Avantages

- Un seul état est active chaque instant (**Absence ε –transitions**)
- Pour chaque alphabet un seul transition est possible (**déterminisme**)
- **Temps de simulation court \rightarrow temps de reconnaissance court**



- Inconvénients

- Nombre d'état peut être très grand
- Difficulté de passage automatique à partir d'une expression régulière