La transpilation de Fortran vers C

Présentation de **Erwan FALAUX-BACHELOT**

travail réalisé avec Yann MIQUEL–ERDMANN

Introduction

Le Fortran : 🅵
Créé par : IMB en 1954
Utilisé surtout dans les supercalculateurs

Problématique

Comment implémenter une conversion rapide de programmes Fortran en programmes C ?

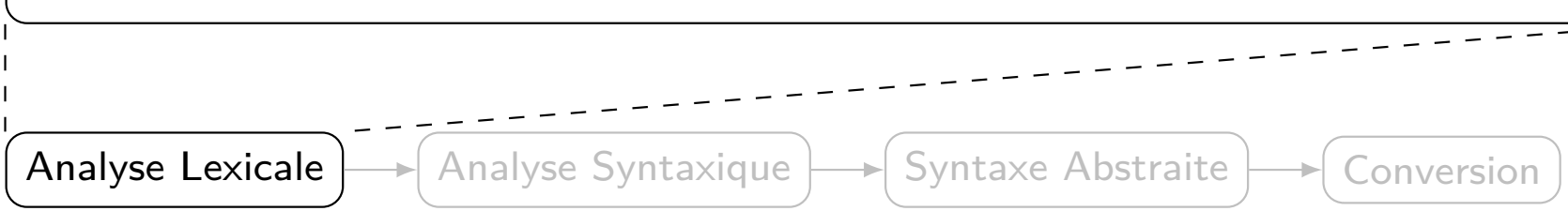
1. Analyse Lexicale
   - Les expressions régulières
   - Les automates
   - La déterminisation
2. Analyse Syntaxique
3. Convertion vers la syntaxe abstraite
4. Traduction vers le langage de sortie

Les expressions régulières

- Expressions régulières
- Automates

définies inductivement sur :

$\emptyset$, ε, $a \in \Sigma$



Les expressions régulières

- Expressions régulières
- Automates

définies inductivement sur :

$$\emptyset,\ \varepsilon,\ a \in \Sigma$$

avec les règles usuelles :

$$\cdot,\ |,\ *$$

Les expressions régulières

- Expressions régulières
- Automates

définies inductivement sur :

$$\emptyset,\ \varepsilon,\ a \in \Sigma$$

avec les règles usuelles :

$$\cdot,\ |,\ *$$

et des additionnelles :

$$+,\ ?,\ [a-z],\ \sim$$

Les expressions régulières

- Expressions régulières
- Automates

exemple :

[0 - 9]+ (.[0 - 9]+) ?

définies inductivement sur :

$\emptyset$, ε, $a \in \Sigma$

avec les règles usuelles :

$\cdot$, $|$, $*$

et des additionnelles :

$+$, ?, $[a - z]$, $\sim$

Les expressions régulières

- Expressions régulières
- Automates

exemple :

[0 - 9]+ (.[0 - 9]+) ?

définies inductivement sur :

$\emptyset$, ε, $a \in \Sigma$

avec les règles usuelles :

$\cdot$, $|$, $*$

et des additionnelles :

$+$, $?$, $[a - z]$, $\sim$

Les expressions régulières

- Expressions régulières
- Automates

exemple :

[0 - 9]+ (.[0 - 9]+) ?

définies inductivement sur :

$\emptyset$, ε, $a \in \Sigma$

avec les règles usuelles :

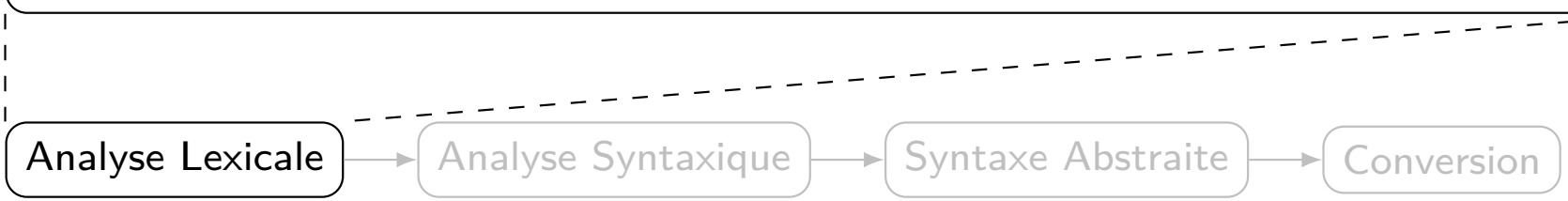
$\cdot$, $|$, $*$

et des additionnelles :

$+$, $?$, $[a - z]$, $\sim$

Les automates

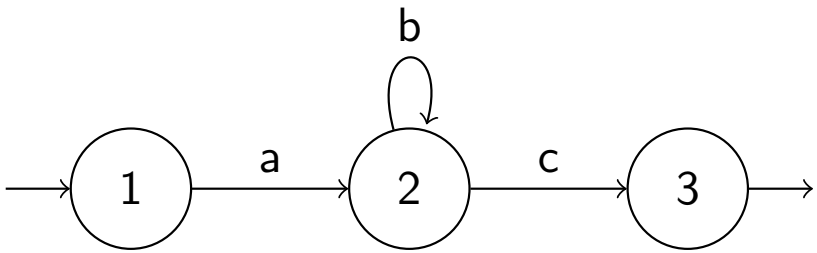
- Expressions régulières
- Automates $(\Sigma, Q, I, F, \delta)$



Les automates

- Expressions régulières
- Automates

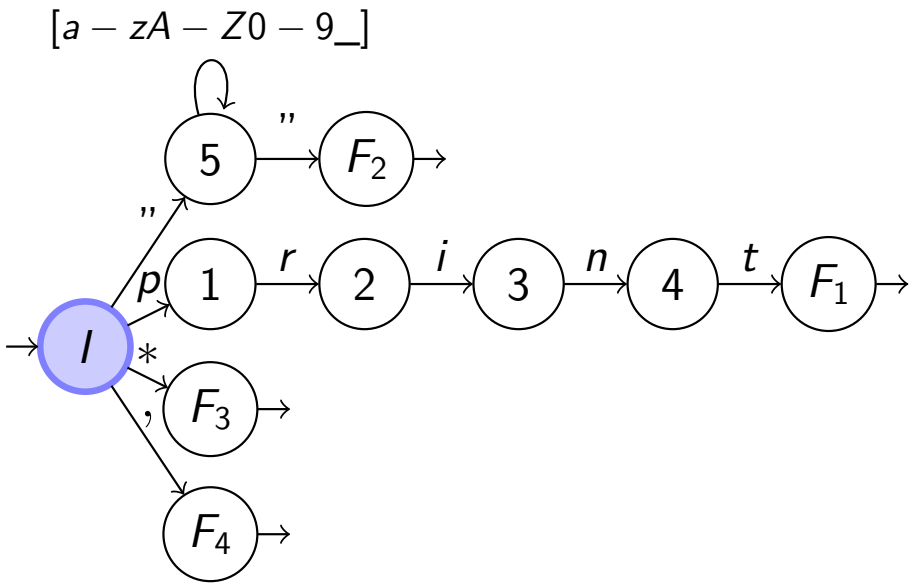
$(\Sigma, Q, I, F, \delta)$



automate pour $a \cdot b* \cdot c$

Analyse Lexicale → Analyse Syntaxique → Syntaxe Abstraite → Conversion

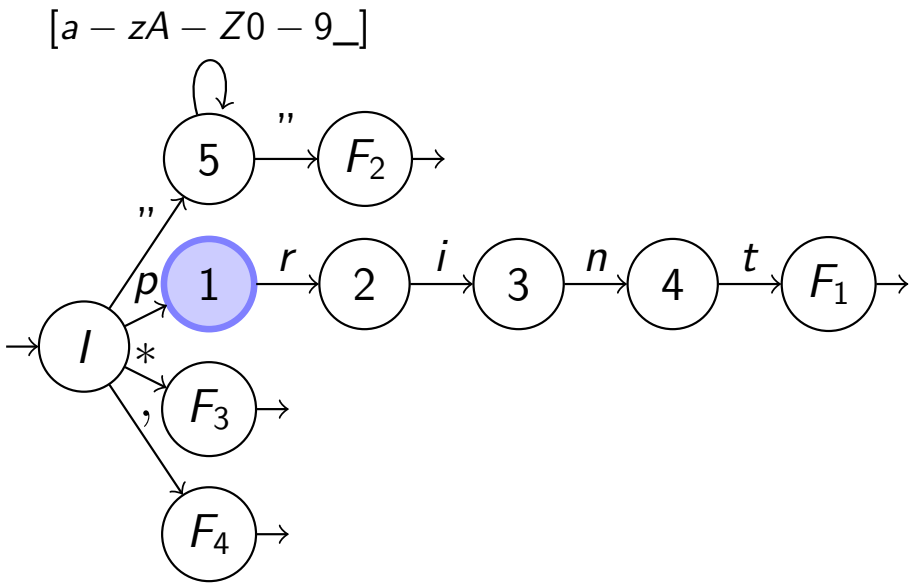
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ ]
```

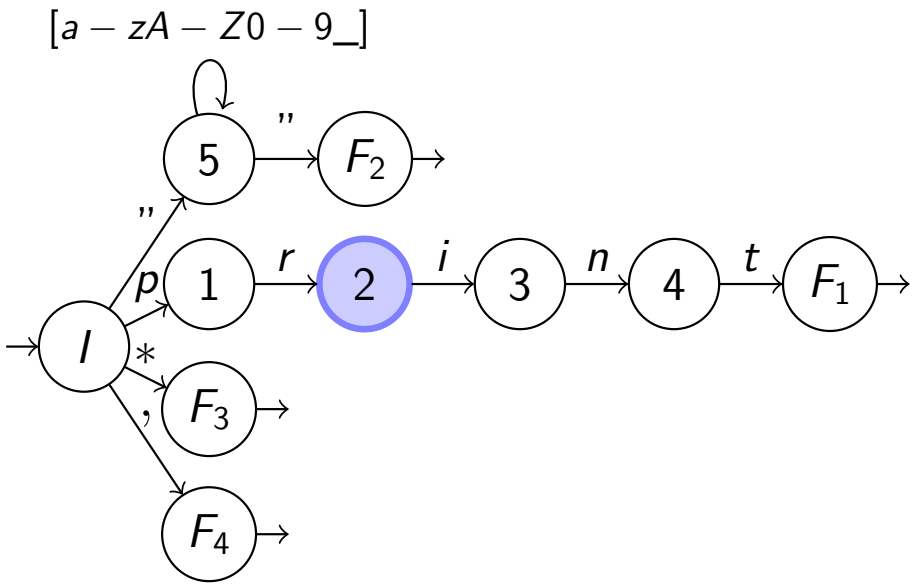
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ ]
```

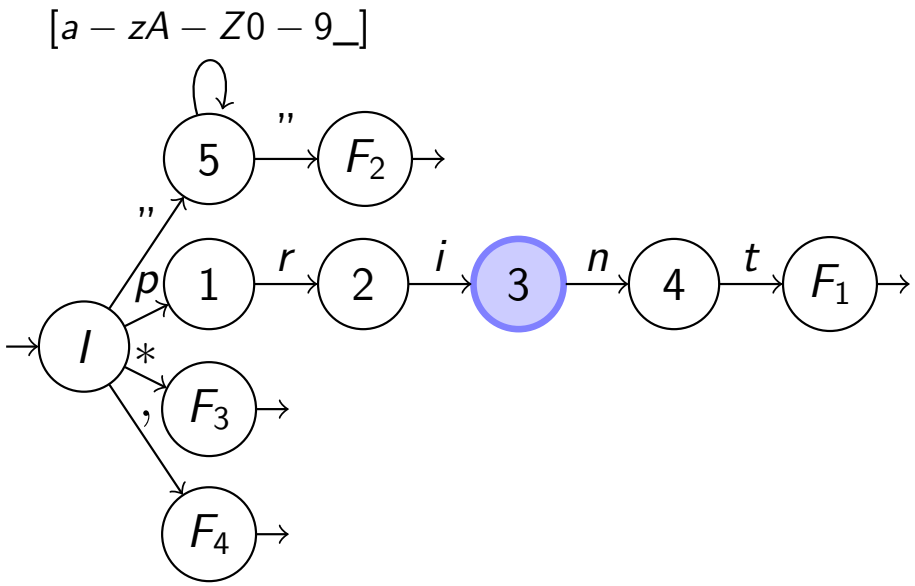
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ ]
```

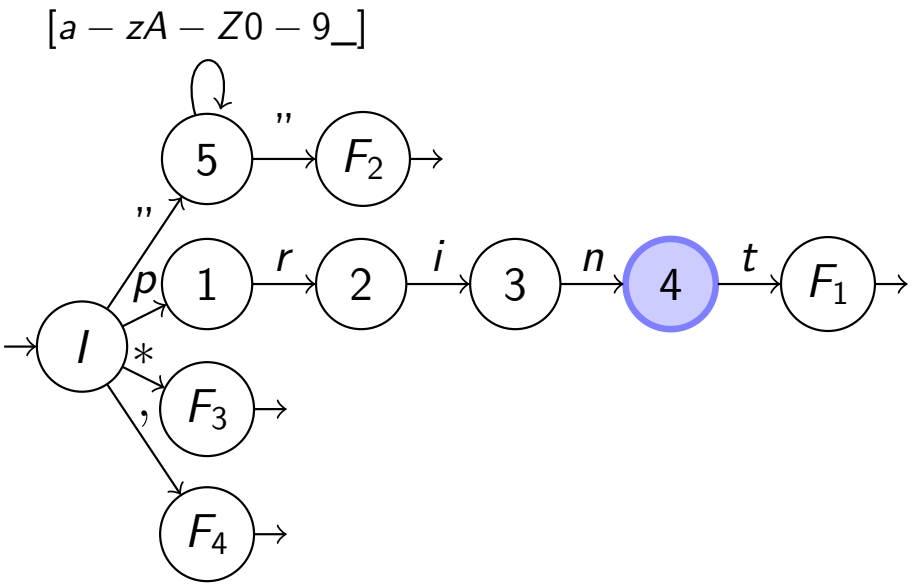
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ ]
```

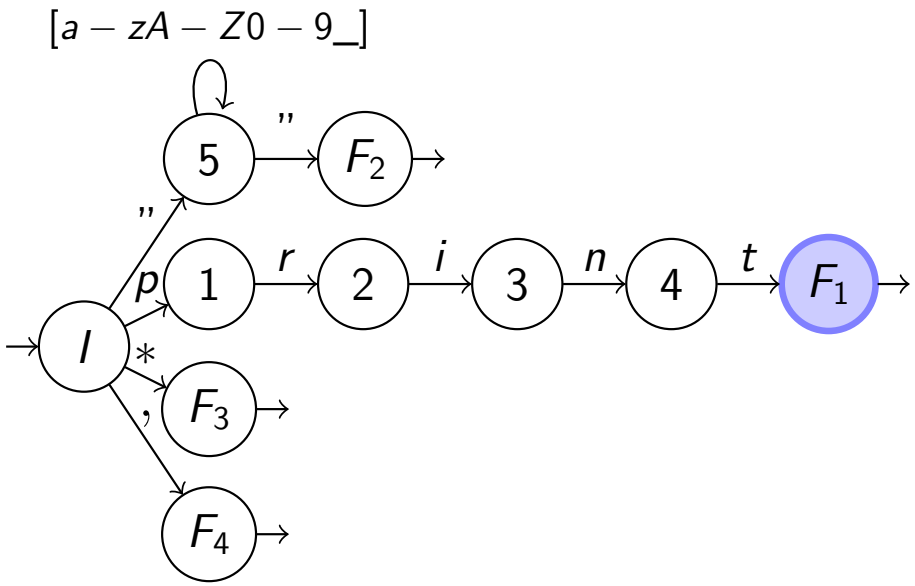
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ ]
```

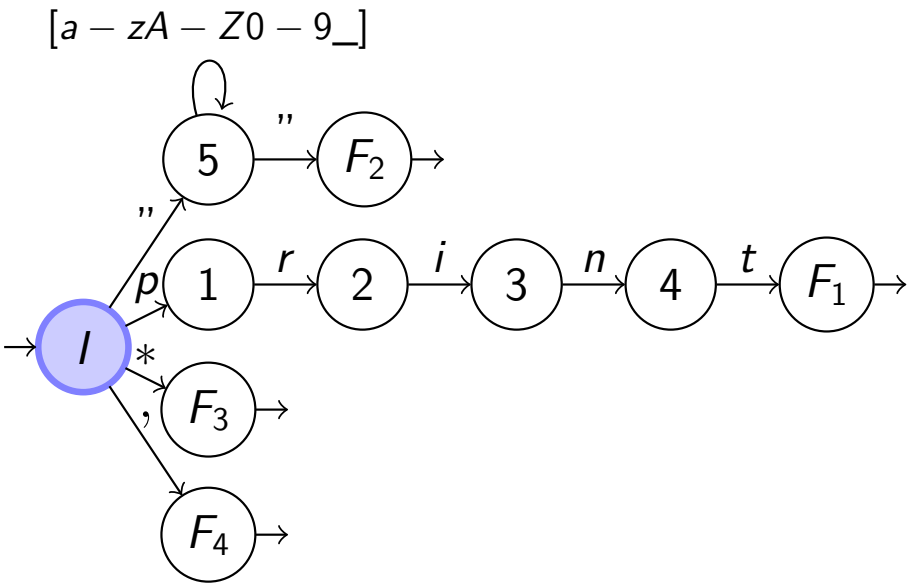
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ ]
```

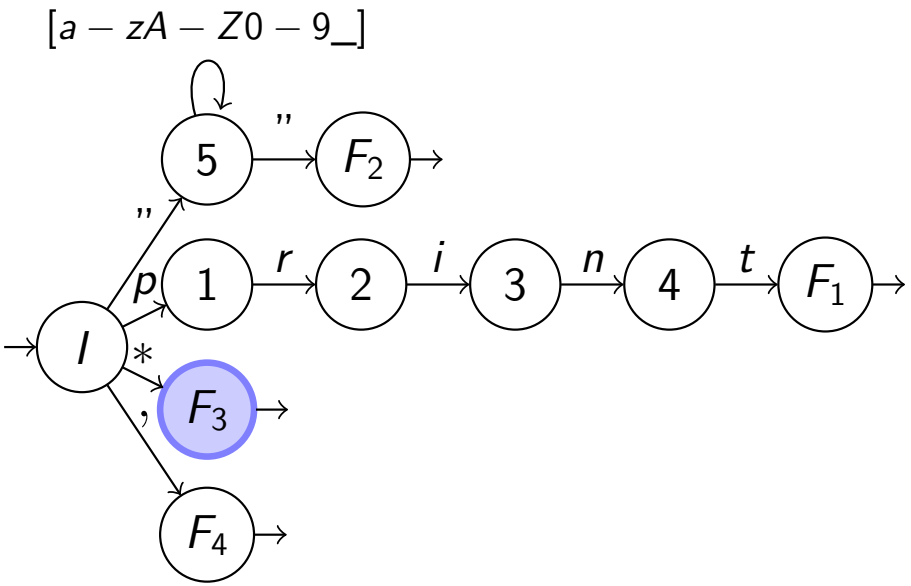
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' ]
```

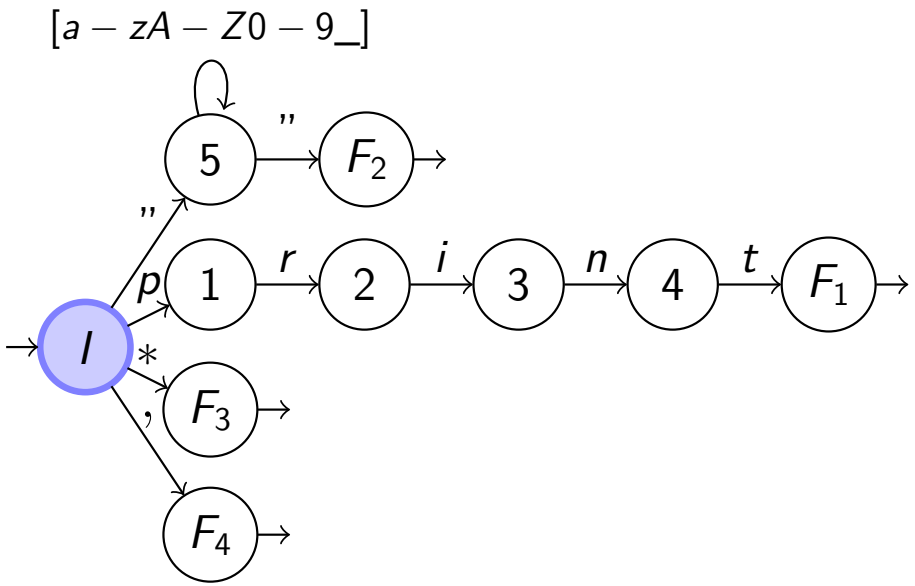
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' ]
```

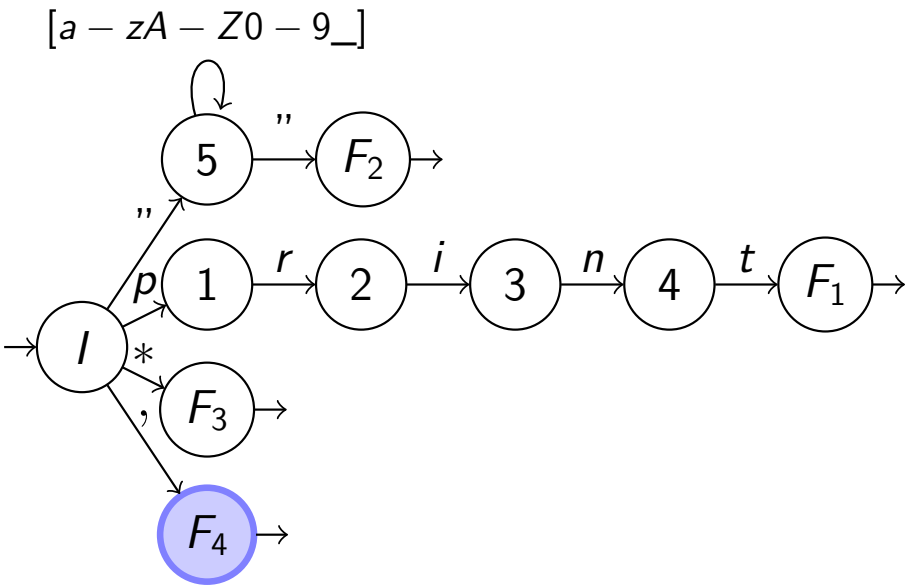
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' ]
```

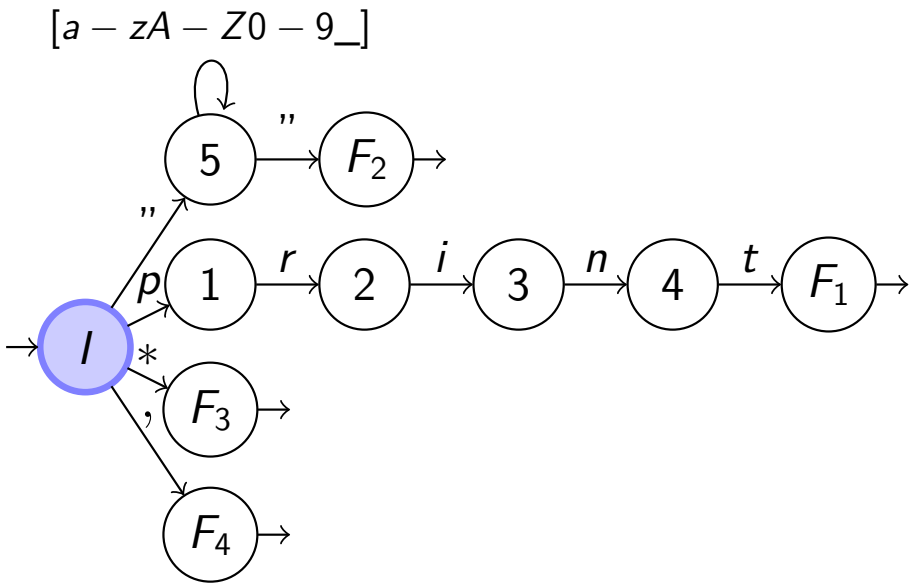
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' ]
```

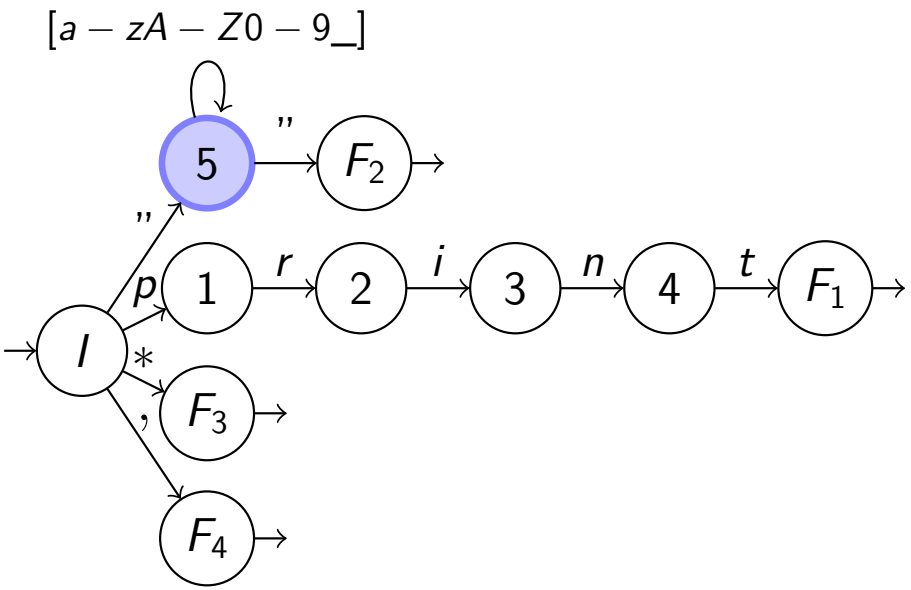
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

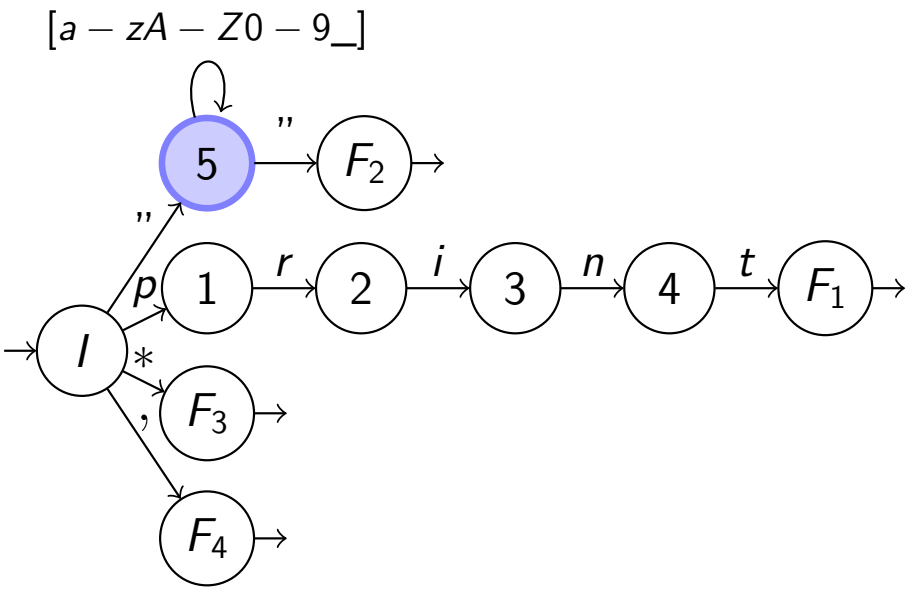
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

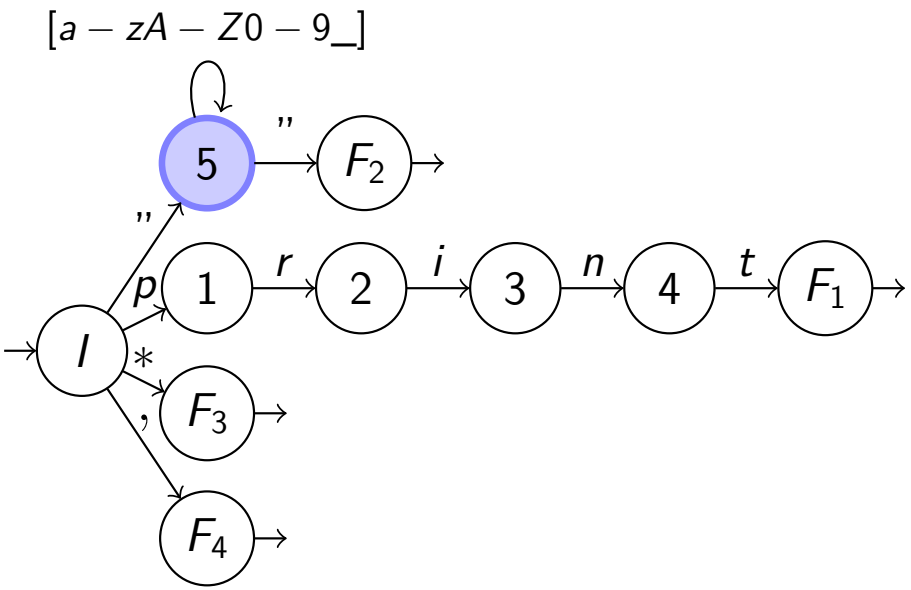
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

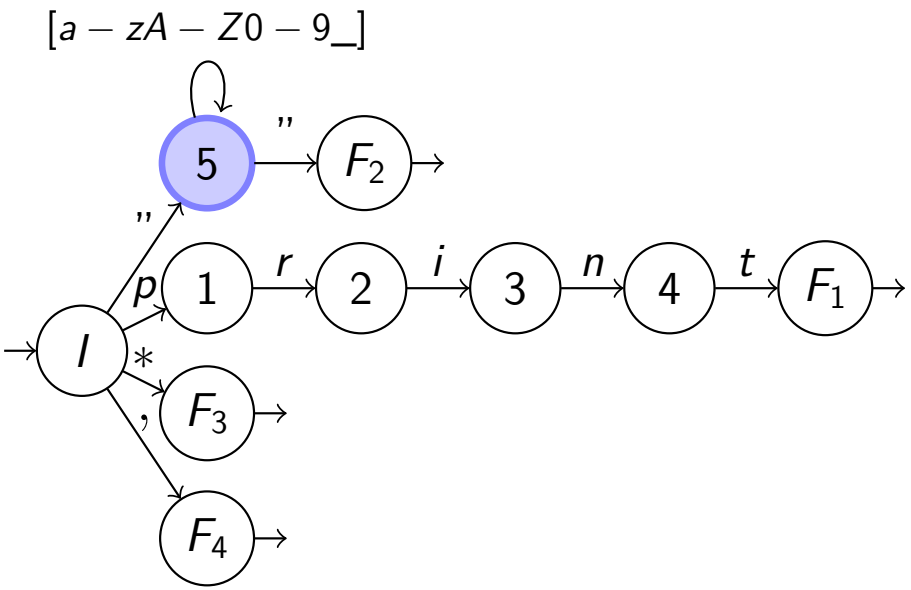
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

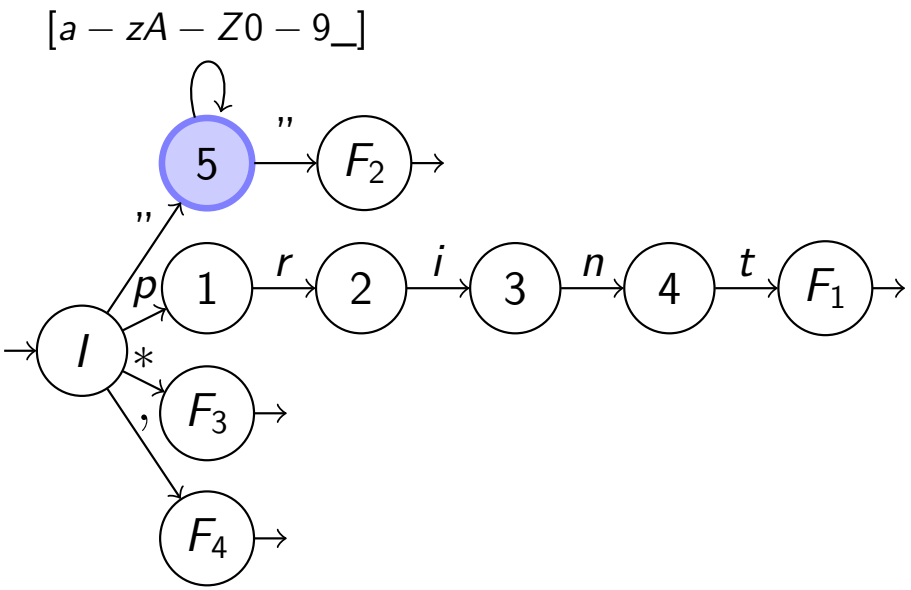
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

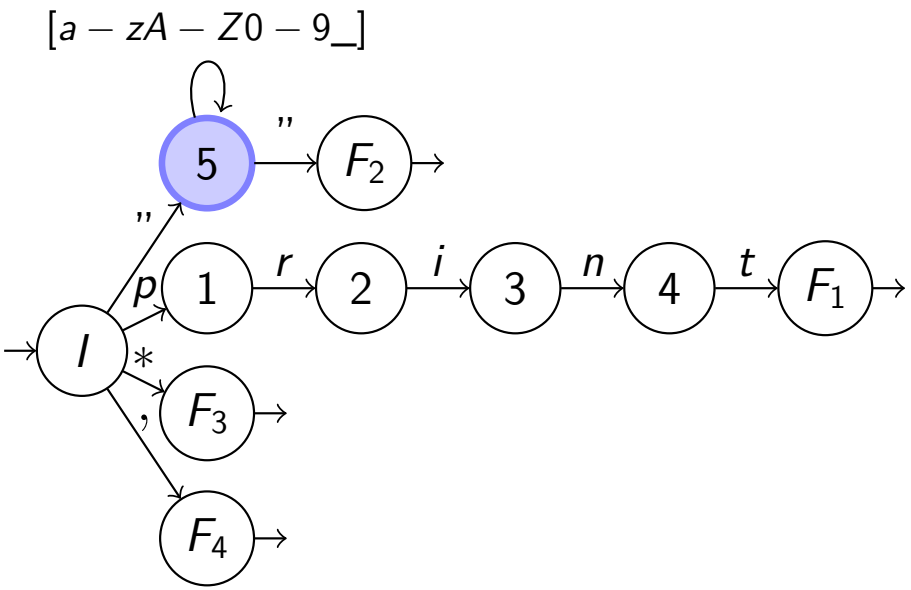
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

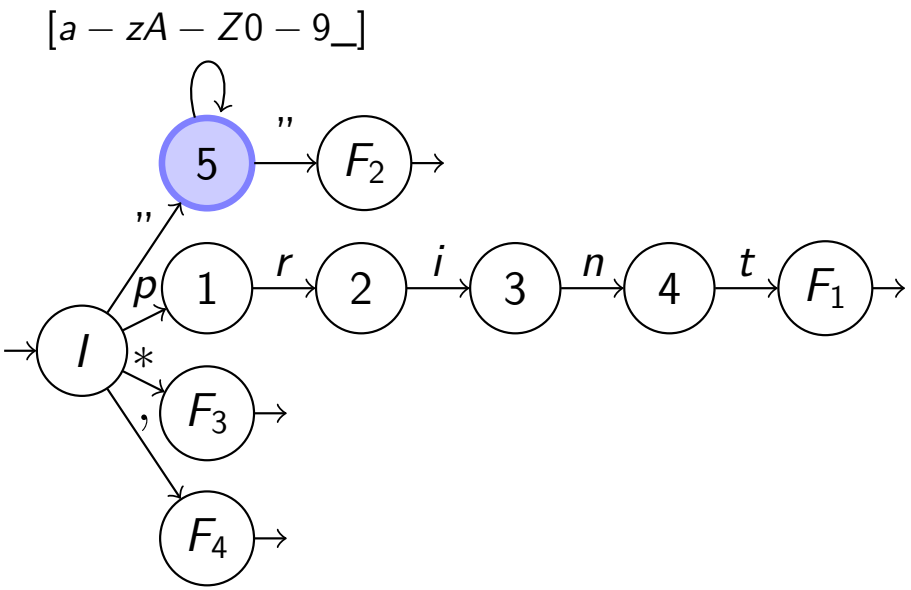
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

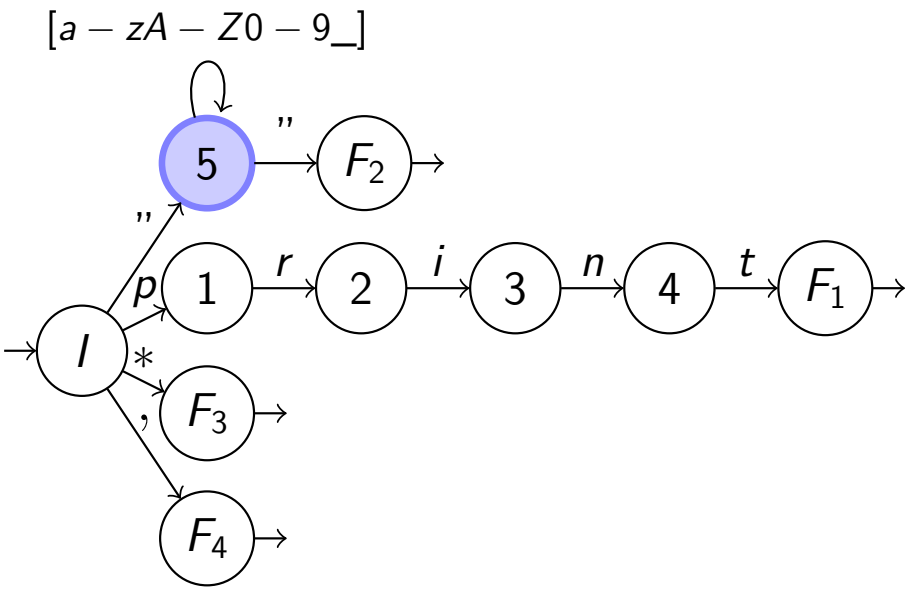
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

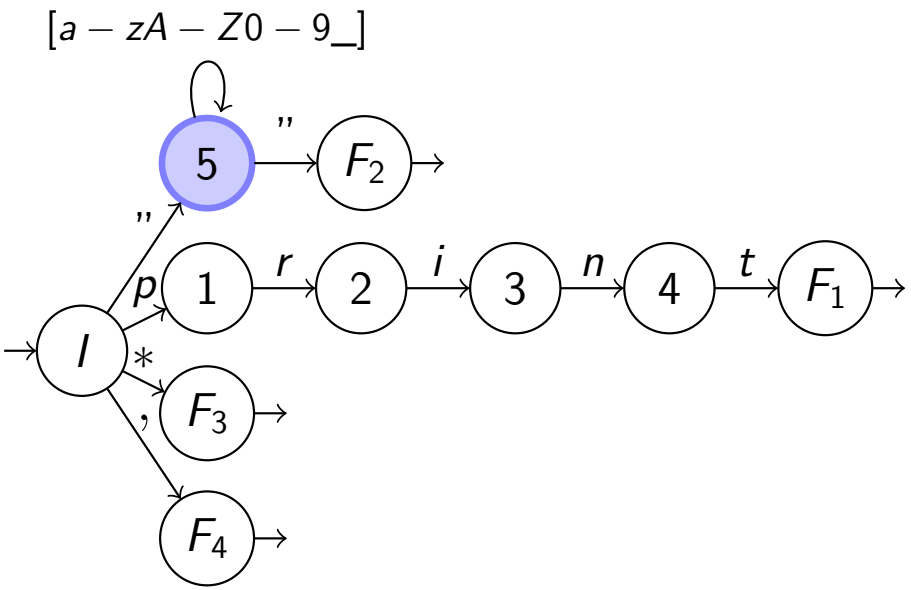
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

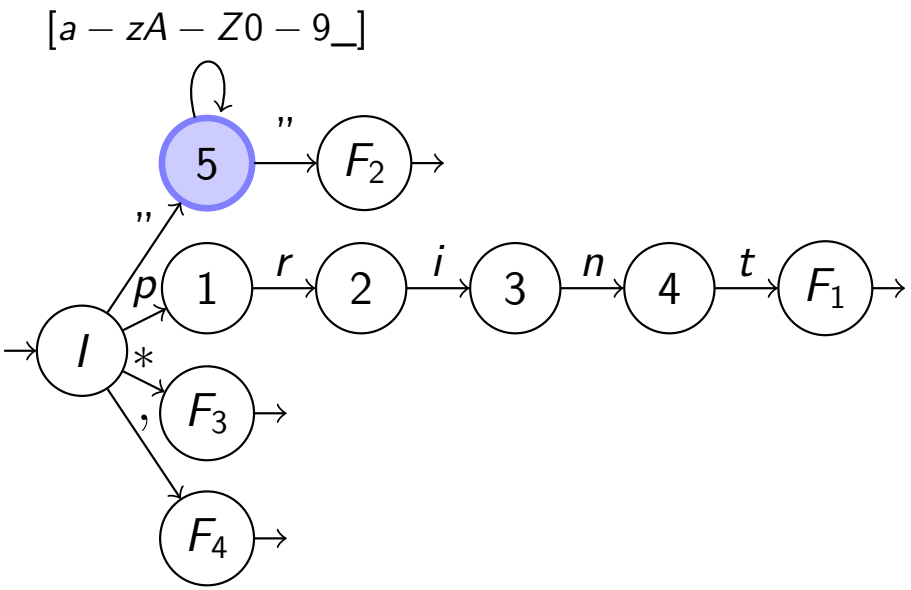
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

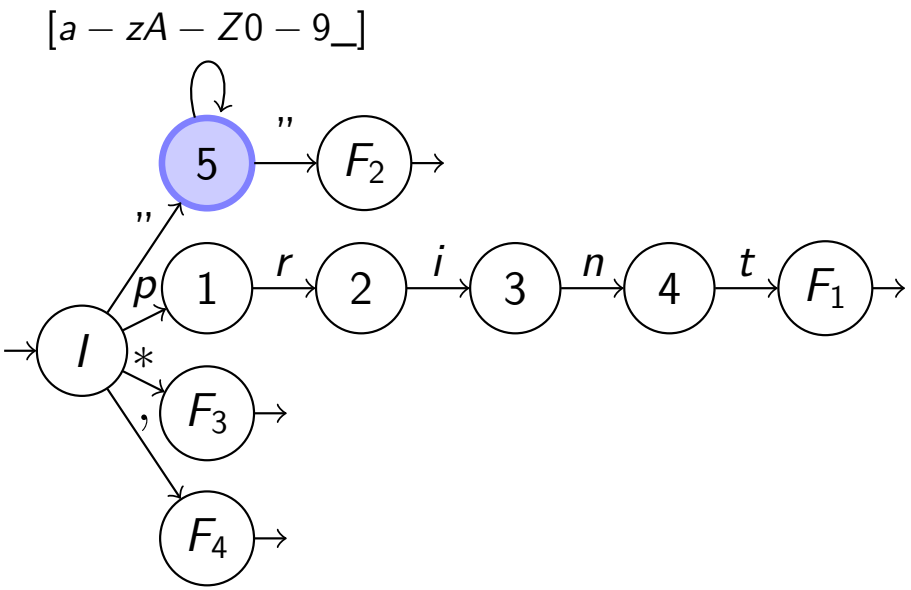
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

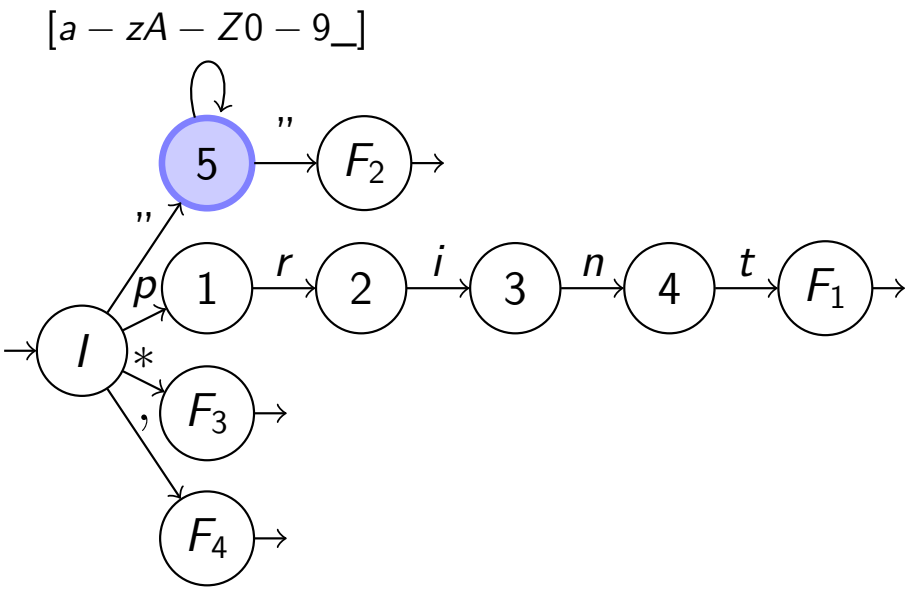
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

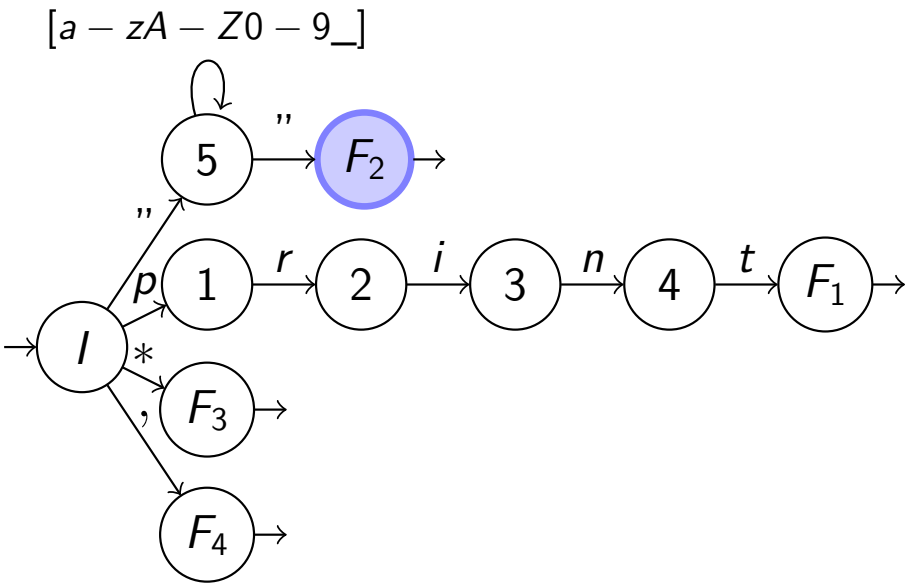
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

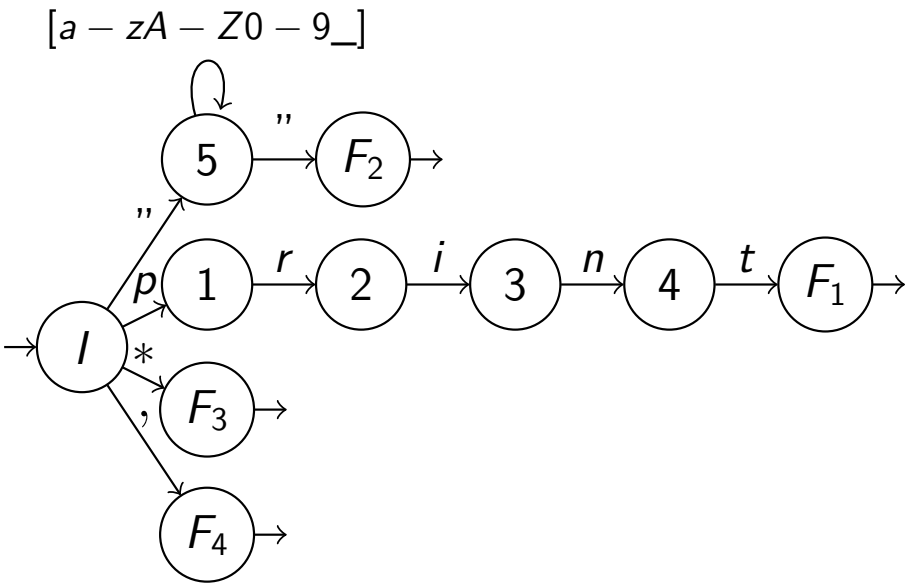
Les automates : étude d'un exemple




```
print*,"Hello World"
```

```
res = [ 'print' , '*' , ',' ]
```

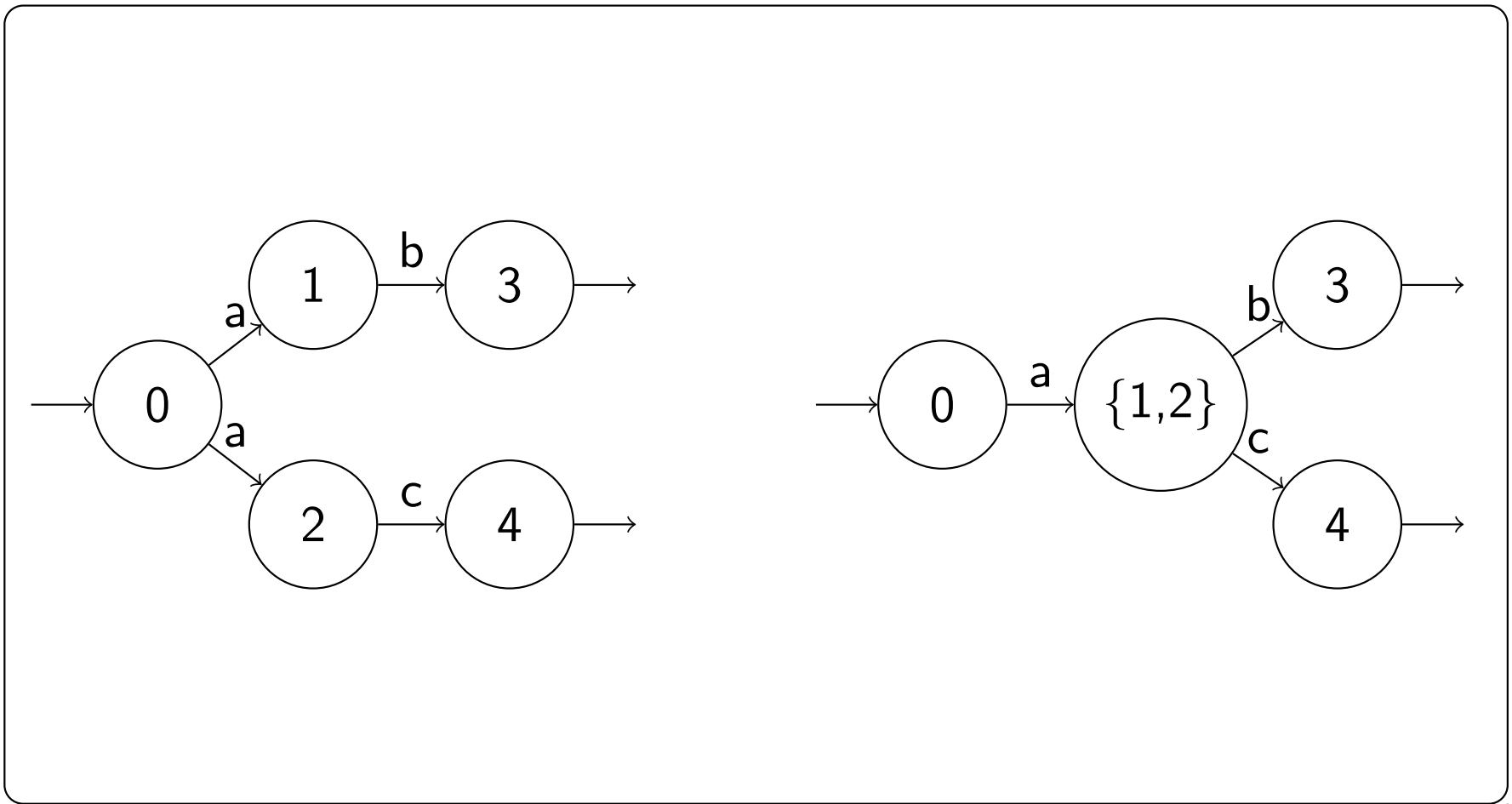
Les automates : étude d'un exemple




```
print*,"Hello World"
```

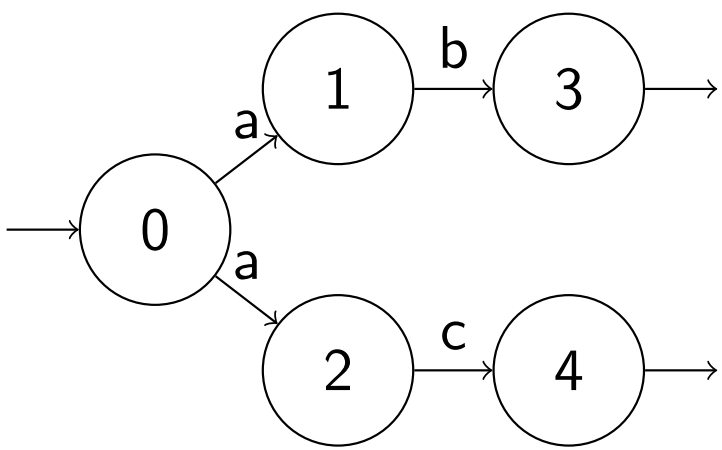
```
res = [ 'print' , '*' , ',' , '"Hello World"' ]
```

La déterminisation



Analyse Lexicale → Analyse Syntaxique → Syntaxe Abstraite → Conversion

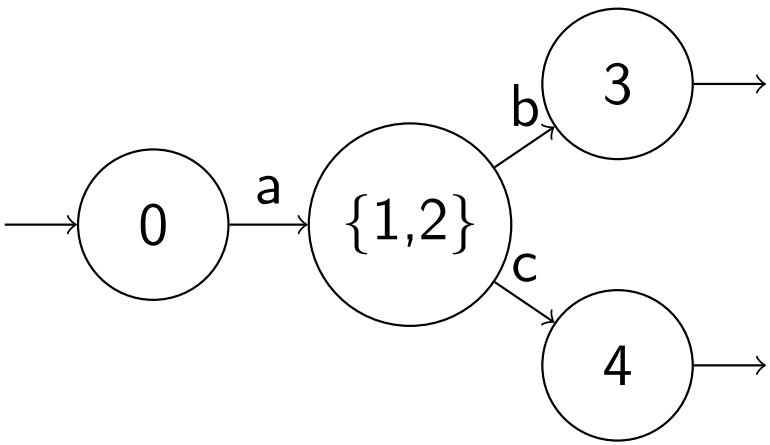
La déterminisation




```
1 type automate = {
2   nodes : int list;
3   debut_l : int list;
4   fin : (int * terminal)
    ↪  list;
5   transitions : (char
    ↪  option * int) list
    ↪  array;
6 }
```

La déterminisation

```
1 type automate_det = {
2   nodes : int list;
3   debut : int;
4   fin : terminal option
    ↪  array;
5   transitions : int array
    ↪  array;
6 }
```



1. Analyse Lexicale

2. Analyse Syntaxique
   - La grammaire
   - L'algorithme LL1

3. Convertion vers la syntaxe abstraite

4. Traduction vers le langage de sortie

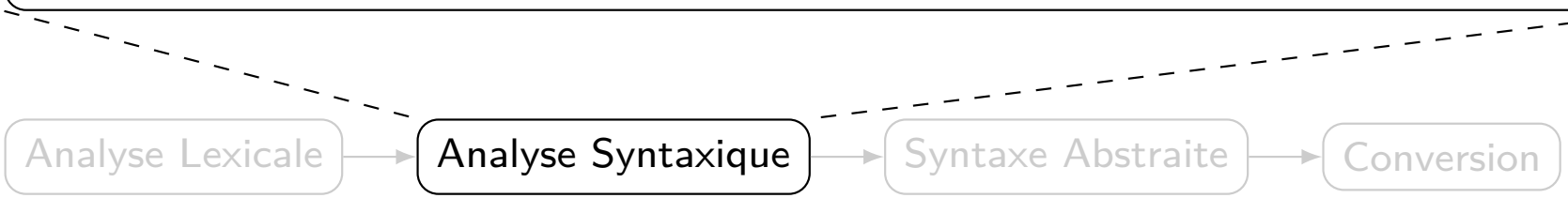
Analyse Syntaxique

- Grammaire
- LL1

Règles de production :

`A -> B`

`A ->` $c, c \in \Sigma^*$



Analyse Syntaxique

- Grammaire
- LL1

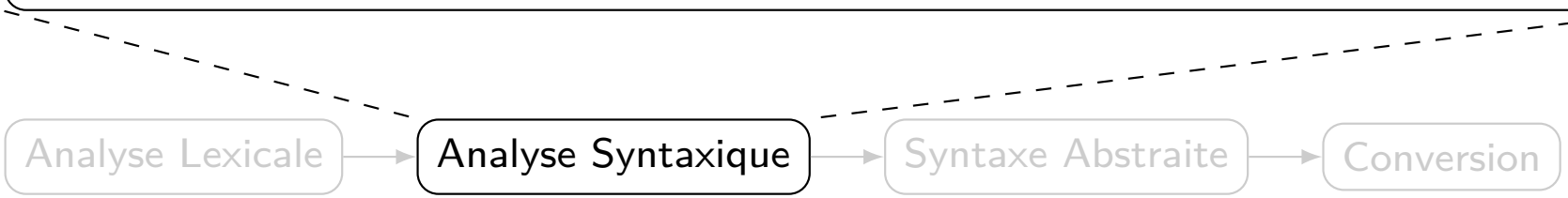
Règles de production :

`A -> B`

`A ->` $c, c \in \Sigma^*$

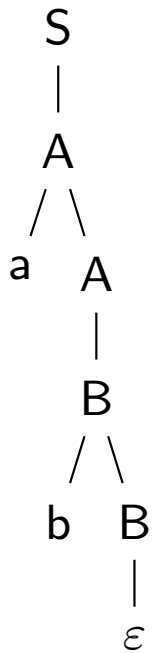
Exemple :

```
S -> A
A -> aA
A -> B
B -> bB
B -> epsilon
```



Analyse Syntaxique

- Grammaire
- LL1



Arbre de dérivation de ab

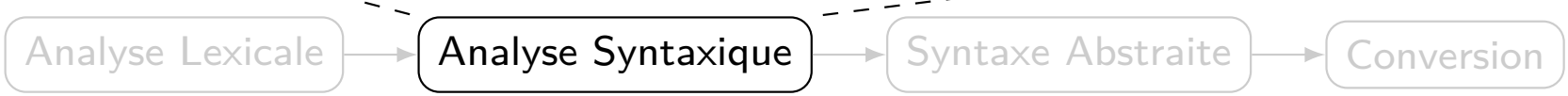
Règles de production :

`A -> B`

`A ->` $c, c \in \Sigma^*$

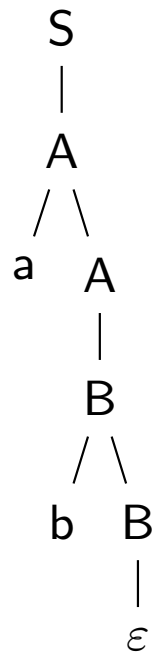
Exemple :

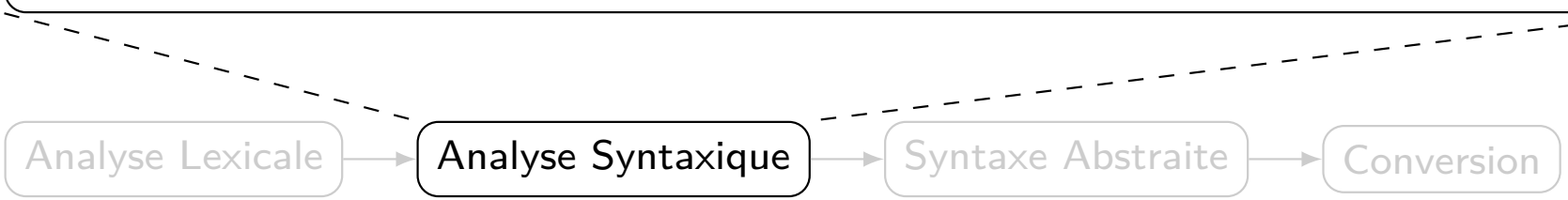
```
S -> A
A -> aA
A -> B
B -> bB
B -> epsilon
```



- Grammaire
- LL1

```
                S -> A
                A -> aA
   [a, b]   +   A -> B          =>
                B -> bB
                B -> epsilon
```





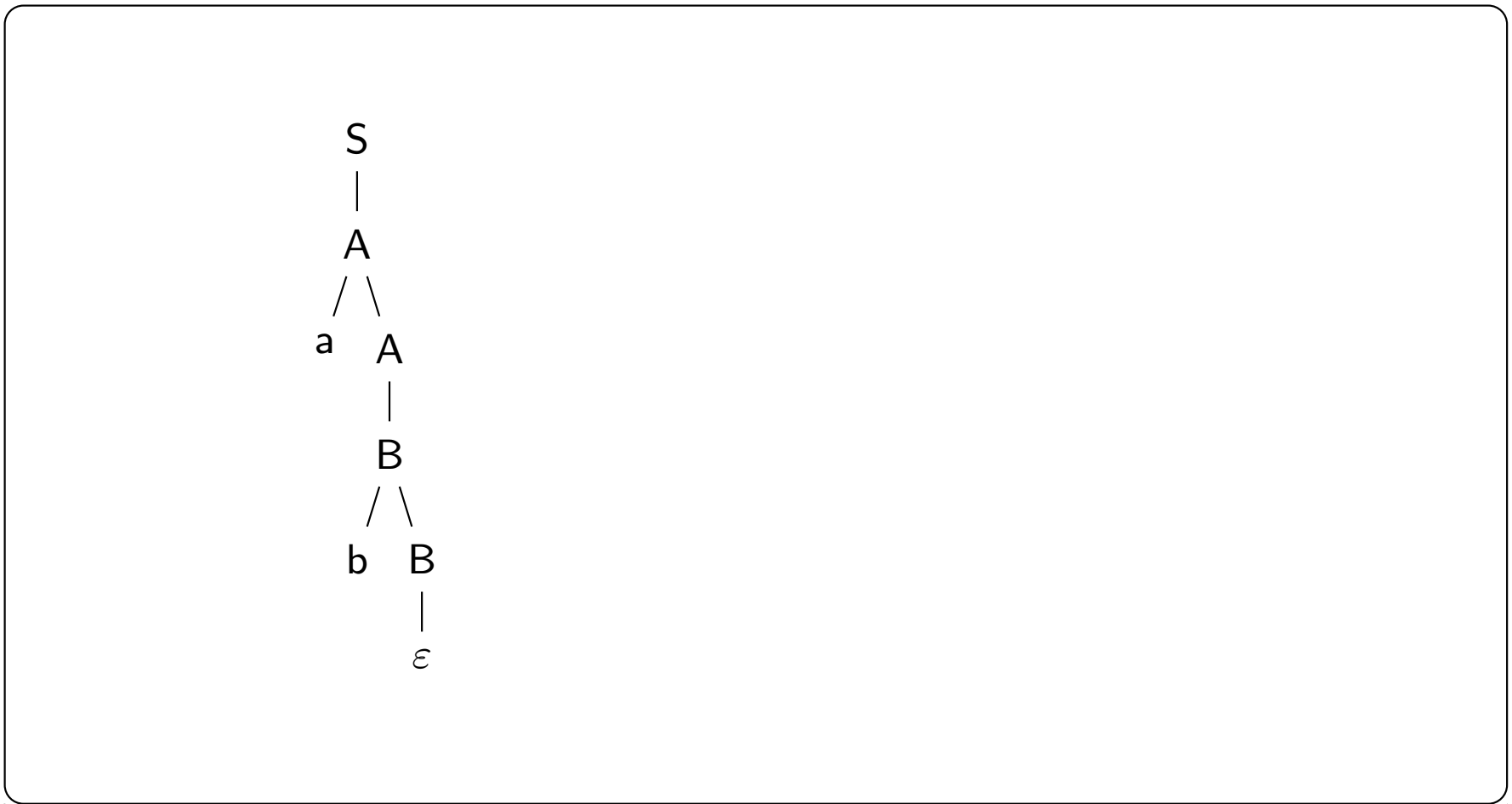
1. Analyse Lexicale

2. Analyse Syntaxique

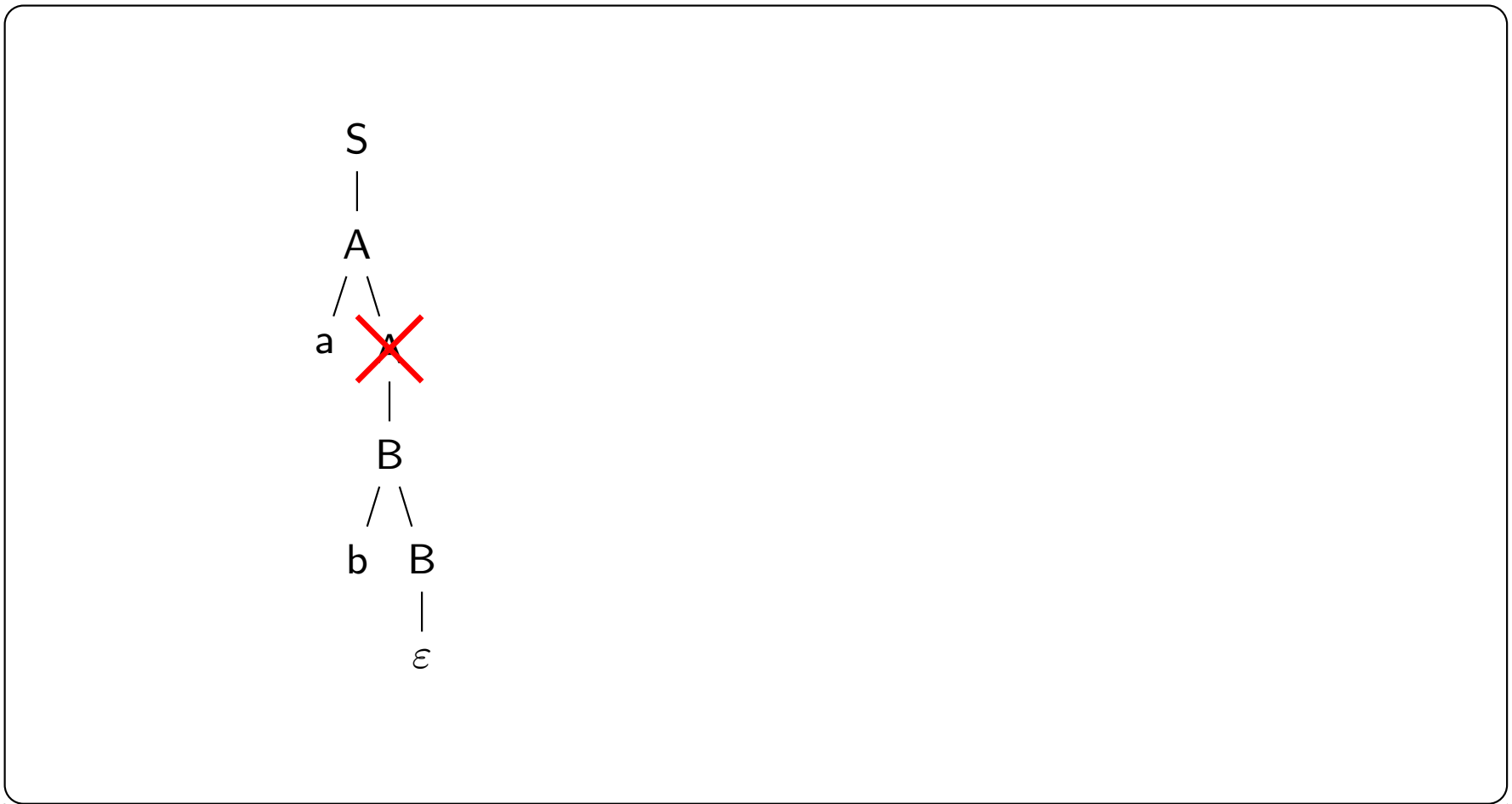
3. Convertion vers la syntaxe abstraite
   - Fonctionnement
   - Un exemple

4. Traduction vers le langage de sortie

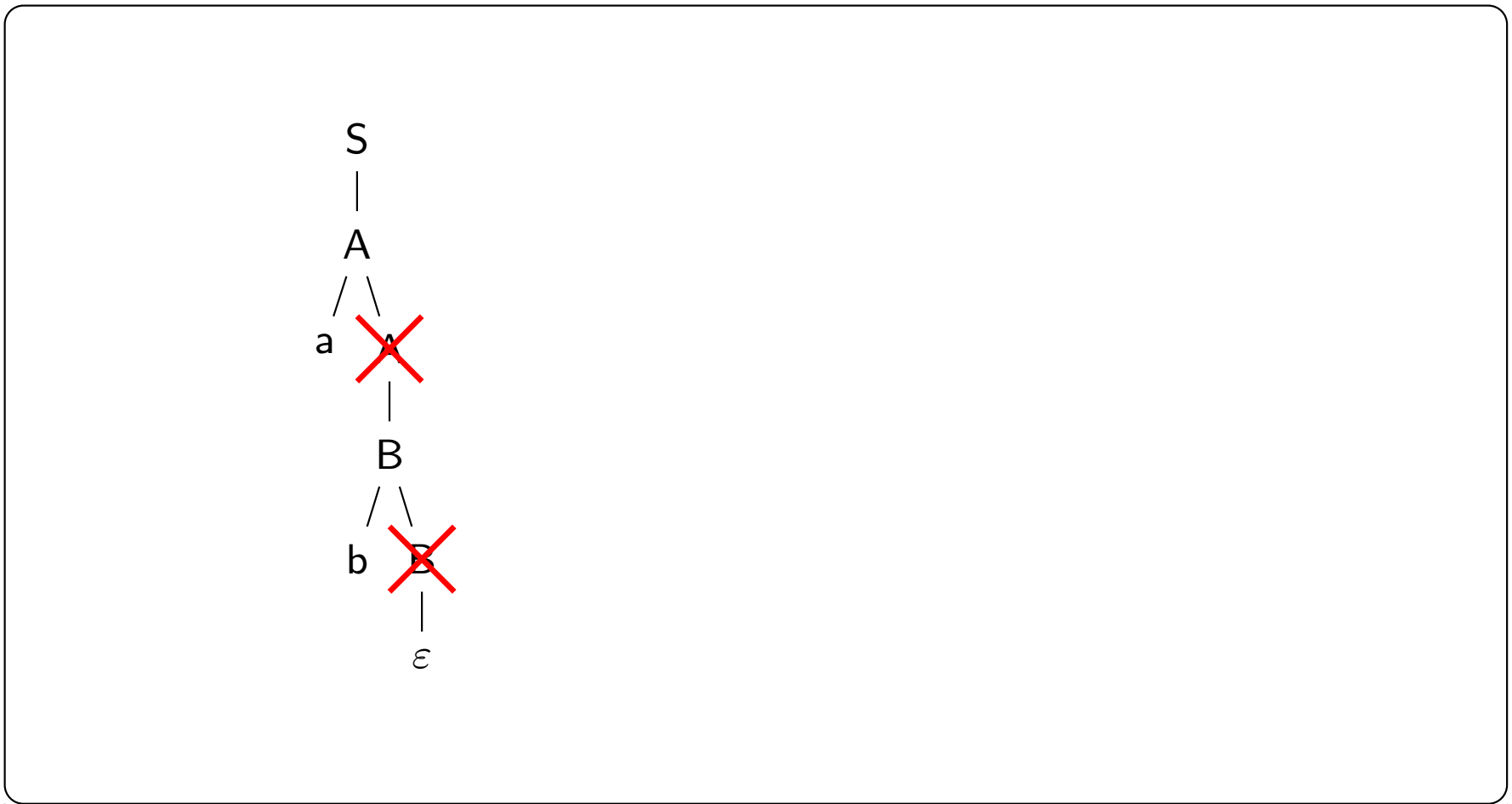
Syntaxe abstraite



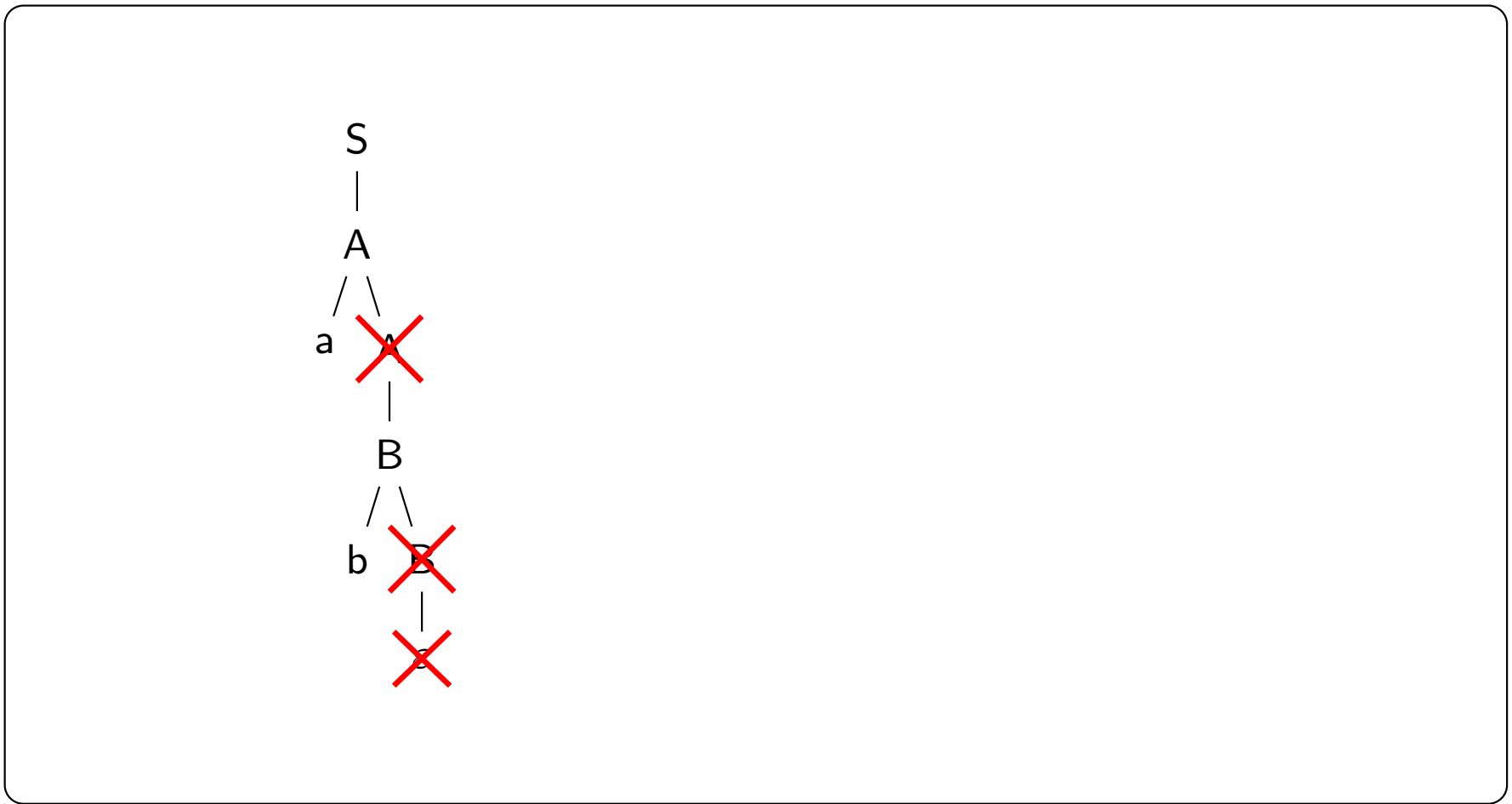
Syntaxe abstraite



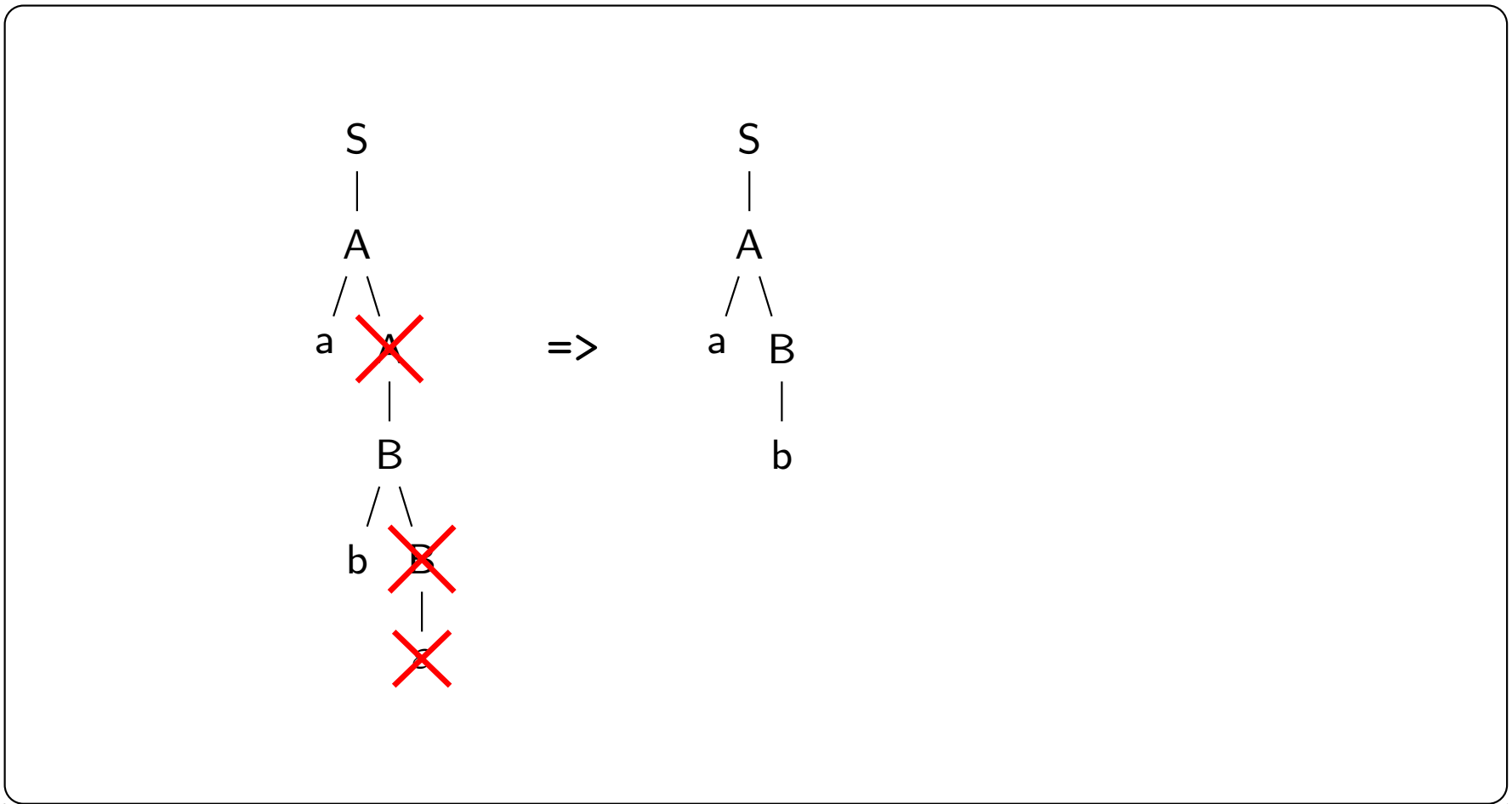
Syntaxe abstraite



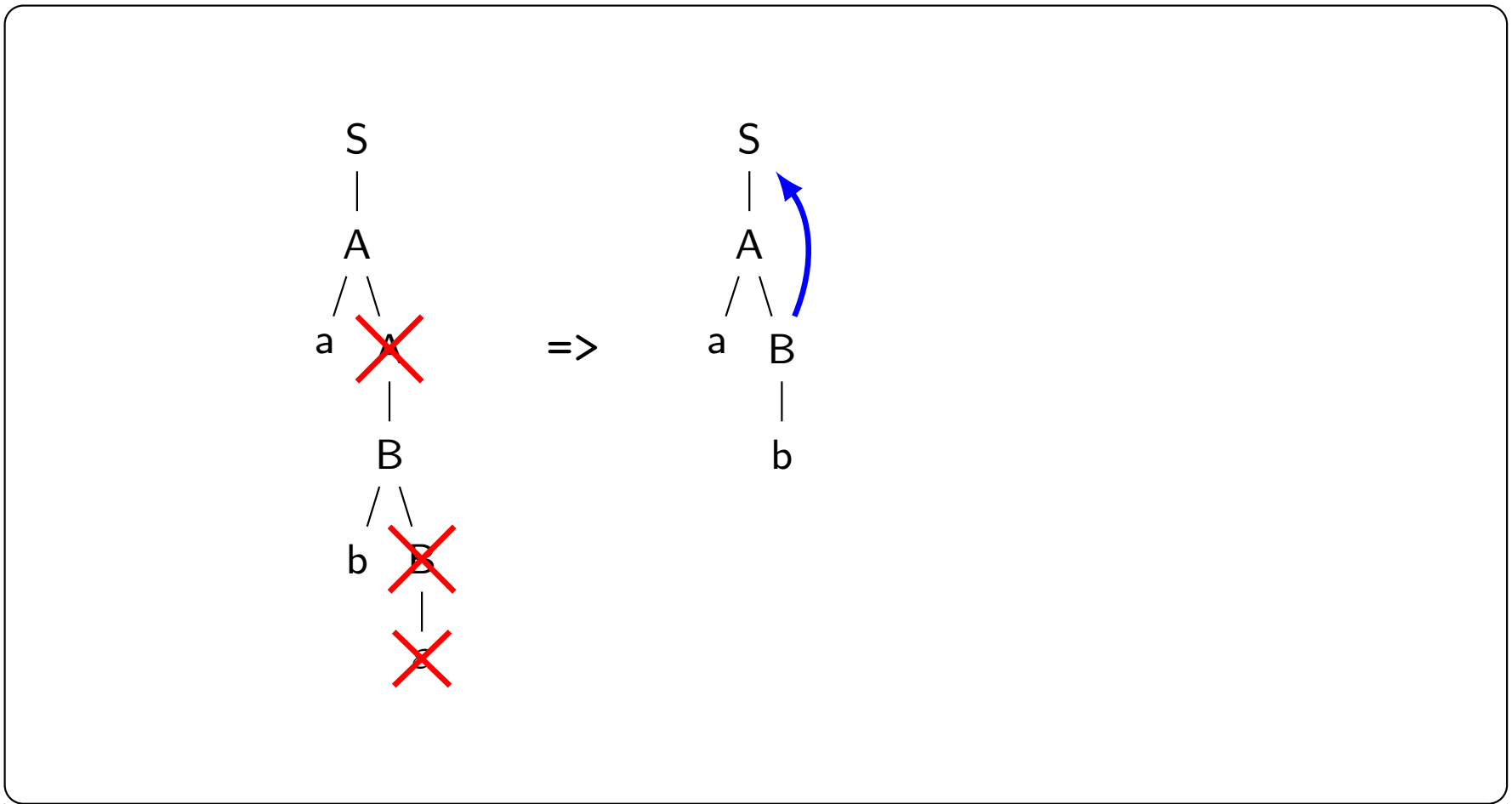
Syntaxe abstraite



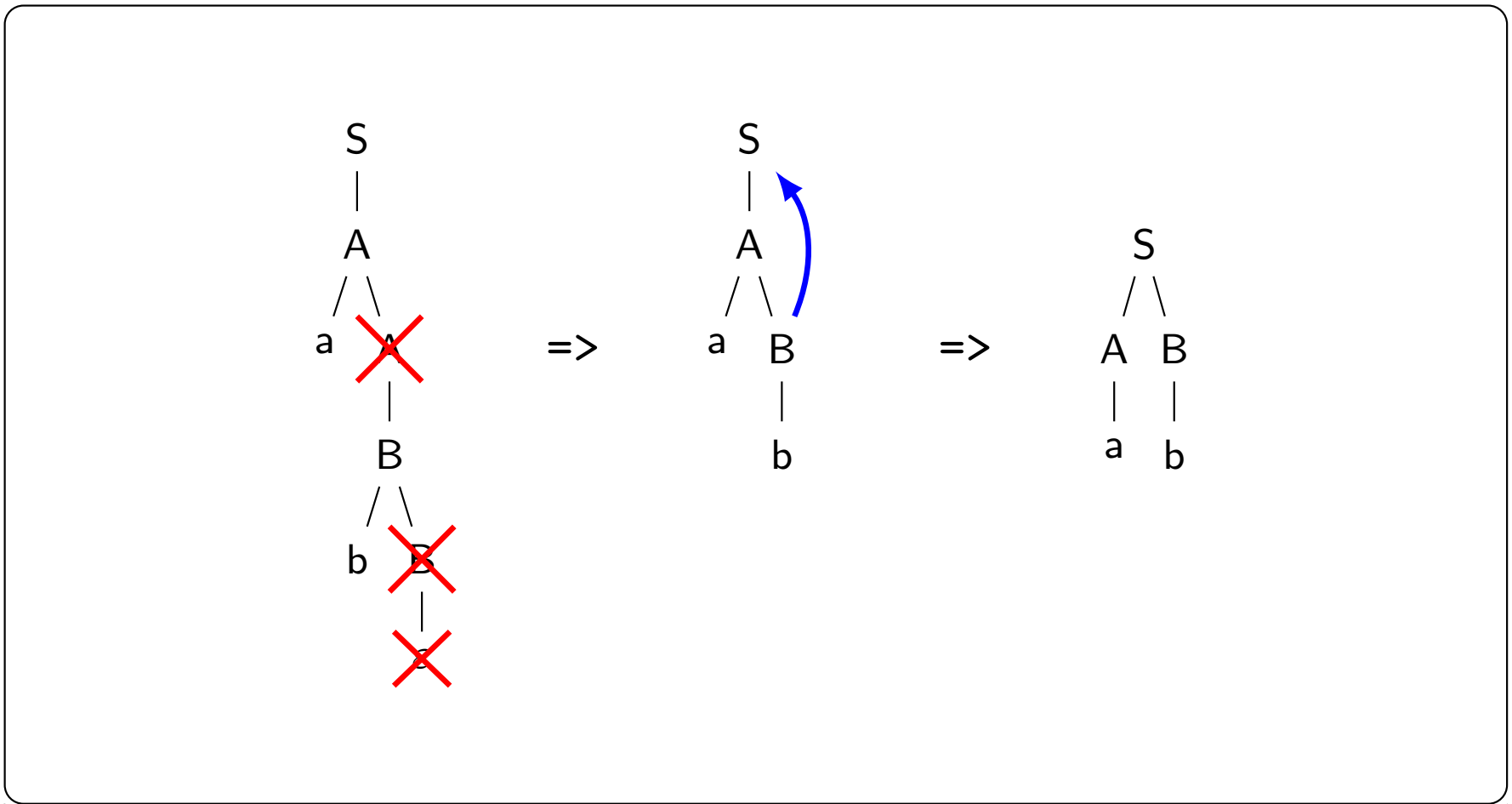
Syntaxe abstraite

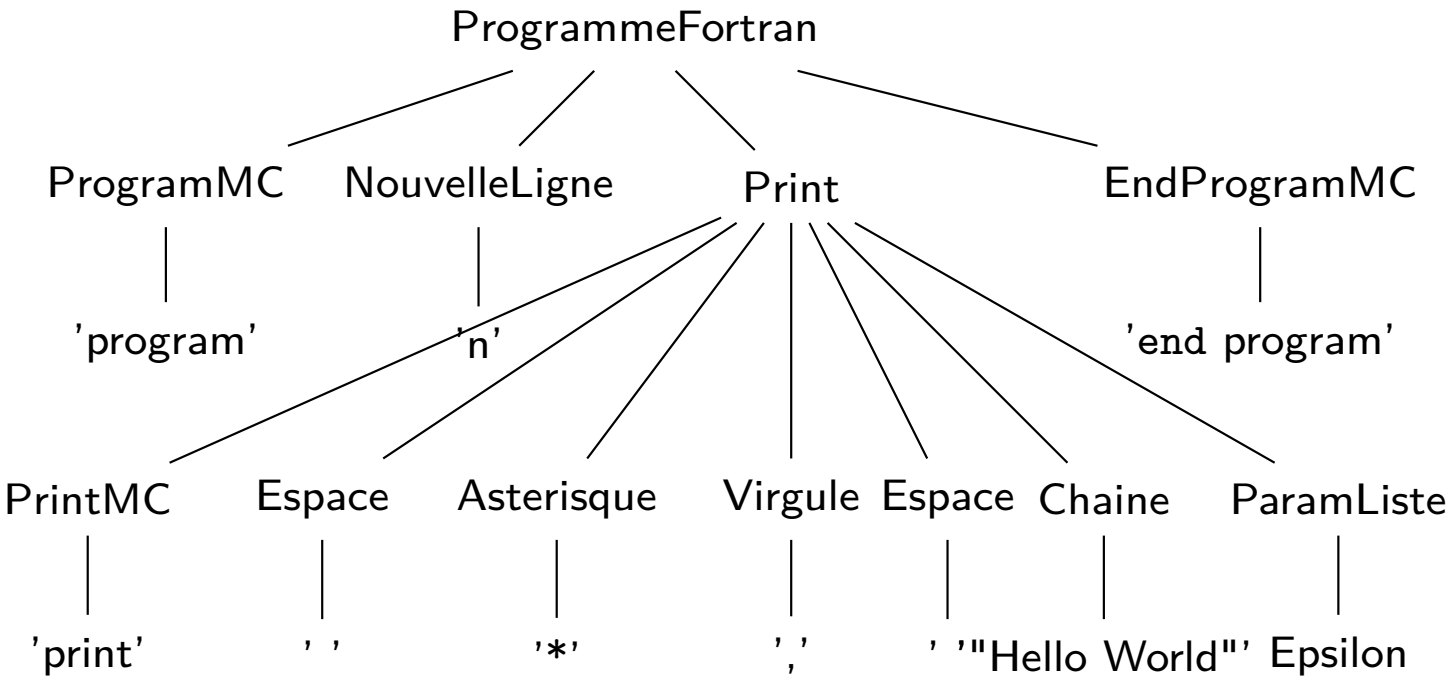


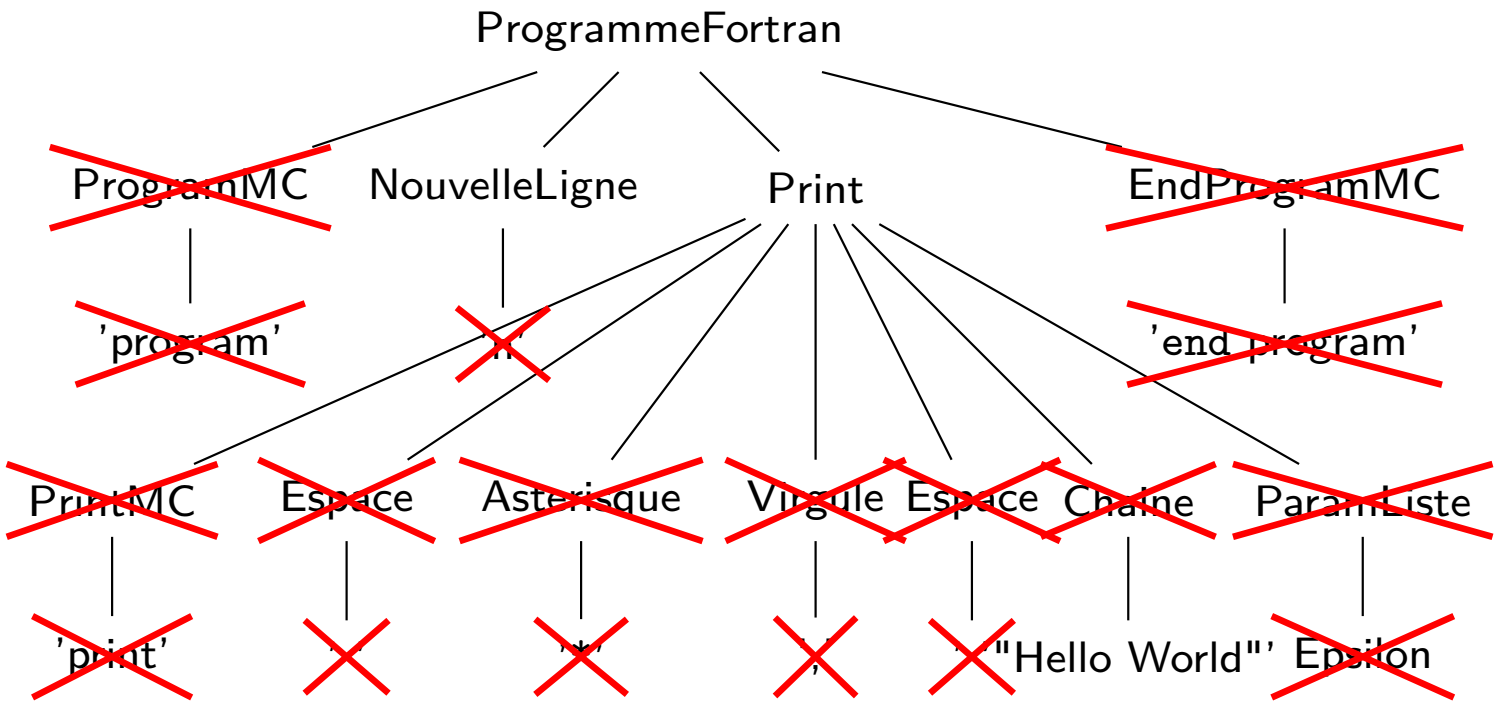
Syntaxe abstraite

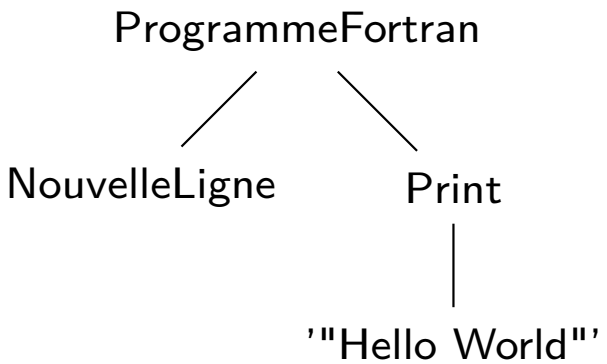


Syntaxe abstraite




ProgrammeFortran
ProgramMC
NouvelleLigne
Print
EndProgramMC
'program'
'n'
'end program'
PrintMC
Espace
Asterisque
Virgule
Espace
Chaine
ParamListe
'print'
' '
'*'
','
' '"Hello World"'
Epsilon


ProgrammeFortran
ProgramMC
NouvelleLigne
Print
EndProgramMC
'program'
n
'end program'
PrintMC
Espace
Asterisque
Virgule
Espace
Chaine
ParamListe
'print'
' '
'*'
','
' '
'"Hello World"'
Epsilon


ProgrammeFortran
NouvelleLigne
Print
'"Hello World"'

1. Analyse Lexicale

2. Analyse Syntaxique

3. Convertion vers la syntaxe abstraite

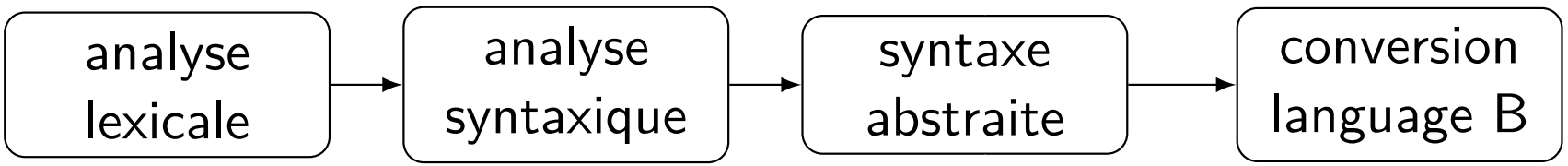
4. Traduction vers le langage de sortie

Traduction

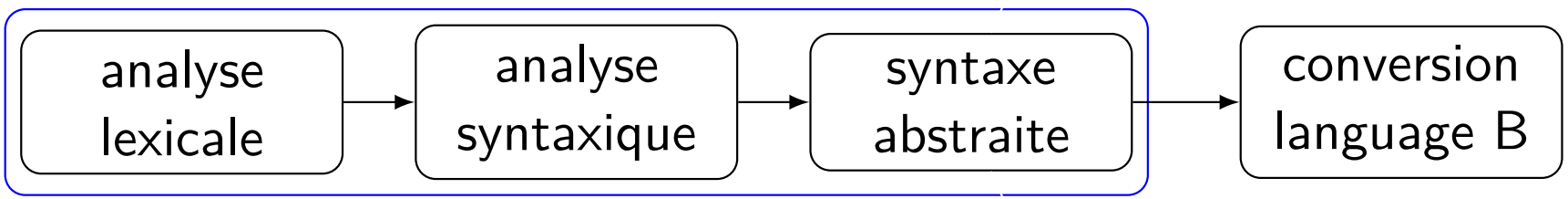
parcours en profondeur

convertion en chaîne

Transpileur

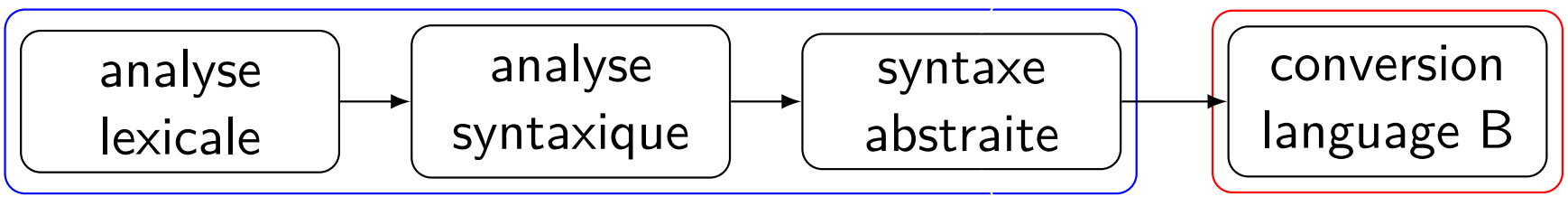


Transpileur



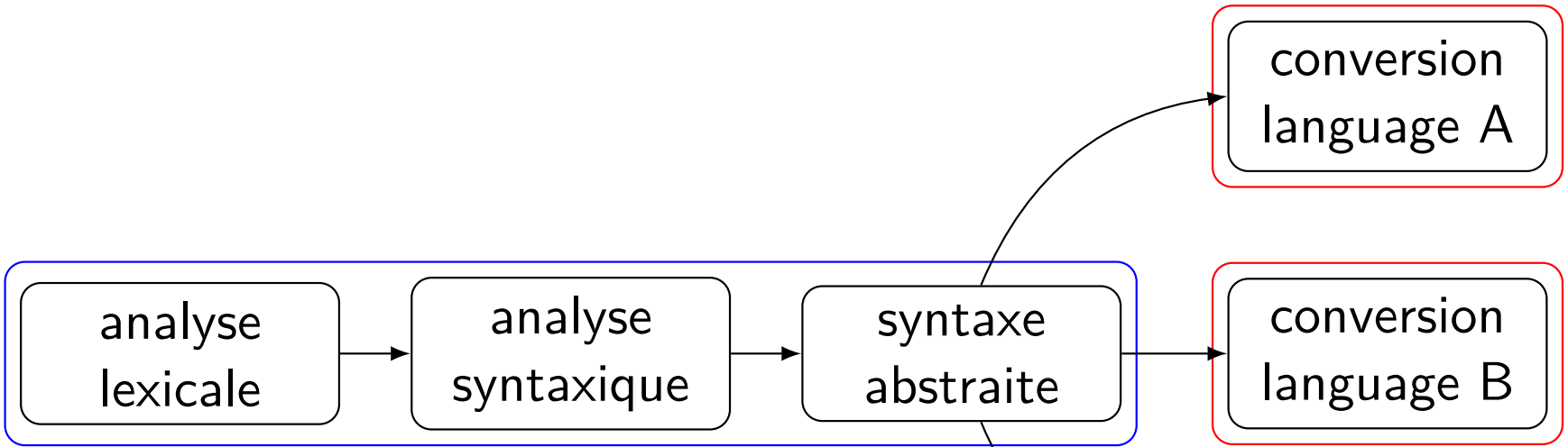
module du langage d'entrée A

Transpileur



module du langage d'entrée A
modules du langage de sortie

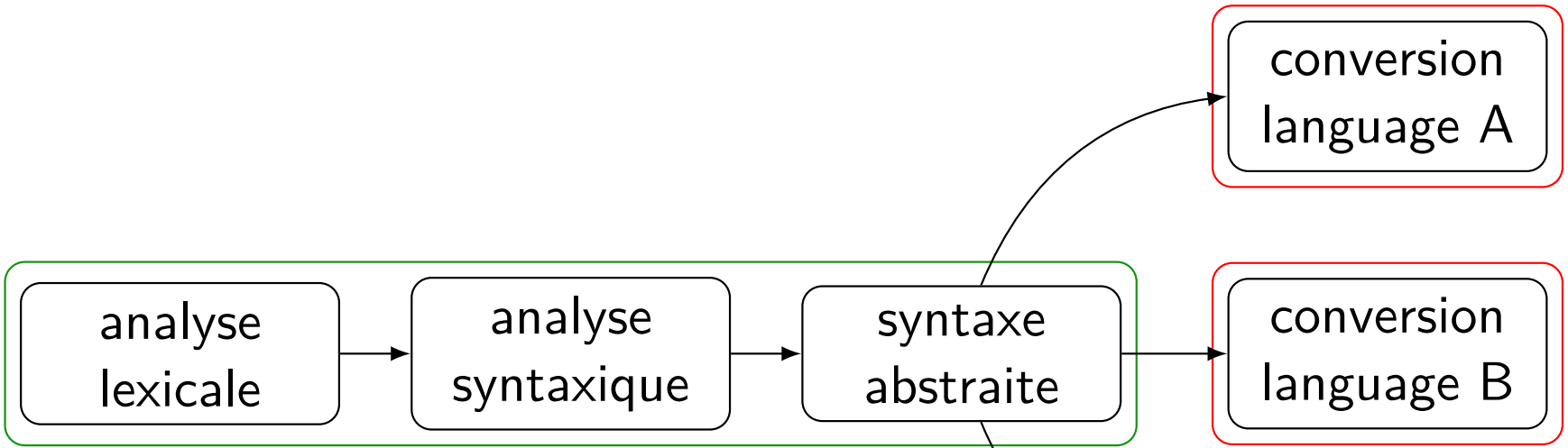
Transpileur



module du langage d'entrée A

modules des langages de sortie

Transpileur



module du langage d'entrée B

modules des langages de sortie

Conclusion

- Conversion rapide
- Partie automatisée rend la création moins pénible
- Processus interchangeable avec les langages souhaités