Expressions régulières/rationnelles

Introduction

- Regular Expressions : RegExp
- Utilisées pour des recherches et des modifications de texte contextuelles avancées
- Grammaires rationnelles / classification Chomsky
- Outils disponibles :
  - Ligne de commandes : `grep, sed, awk`, ..
  - Langages de scripts : `perl, php`, ...
  - Langages de programmation : `java, .NET`
  - Éditeurs de texte : `vi, emacs, jEdit`, ...

➔ Nombreuses variantes de dialecte

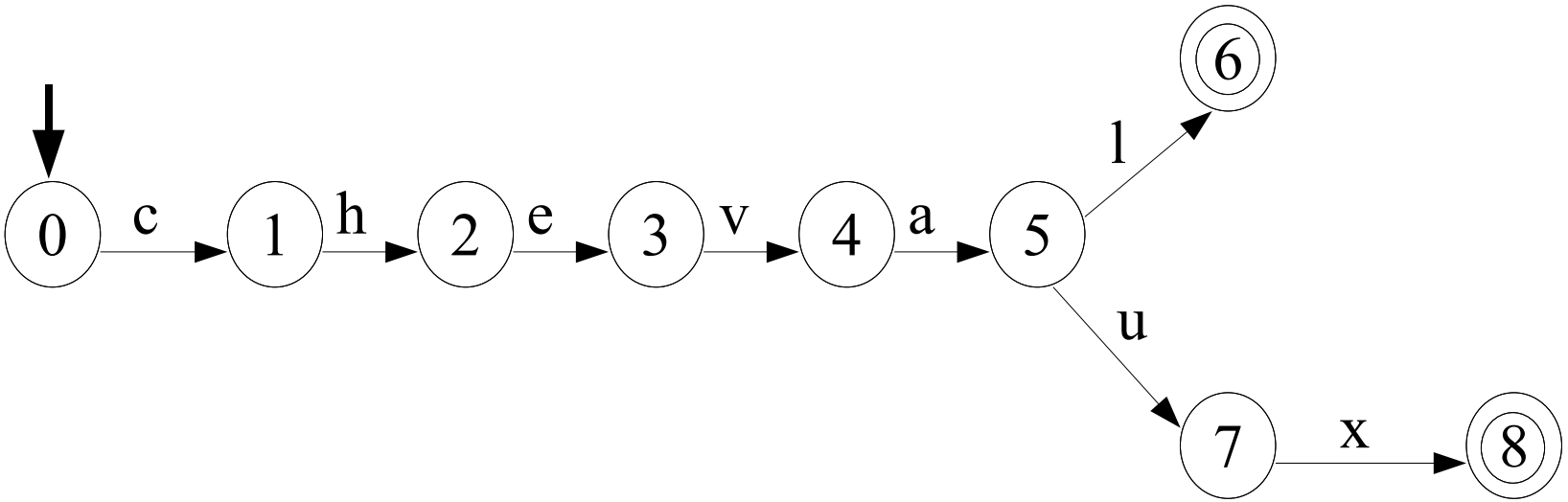
Normes

Deux grandes familles :

- Expression régulière POSIX
- Expression régulière PERL
  - Disponible directement sous PERL
  - Accessible dans d'autres langages avec la librairie PCRE (Perl-Compatible Regular Expressions)

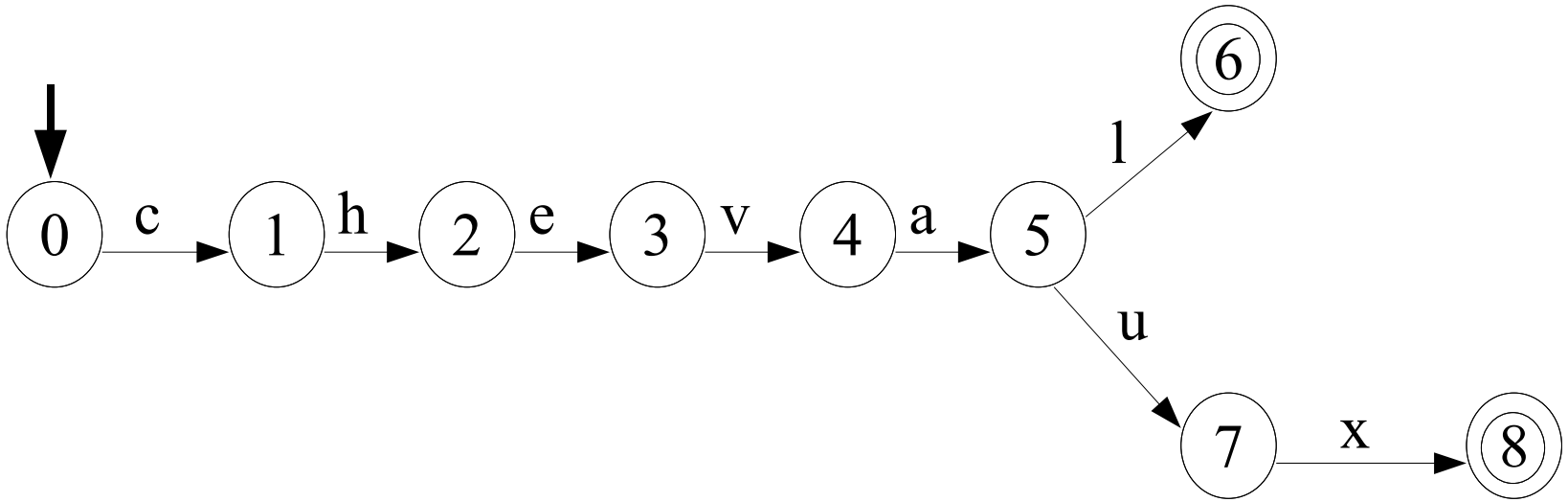
Principes

- Automates finis déterministes (AFD)
  - un état initial
  - un ou plusieurs états finaux



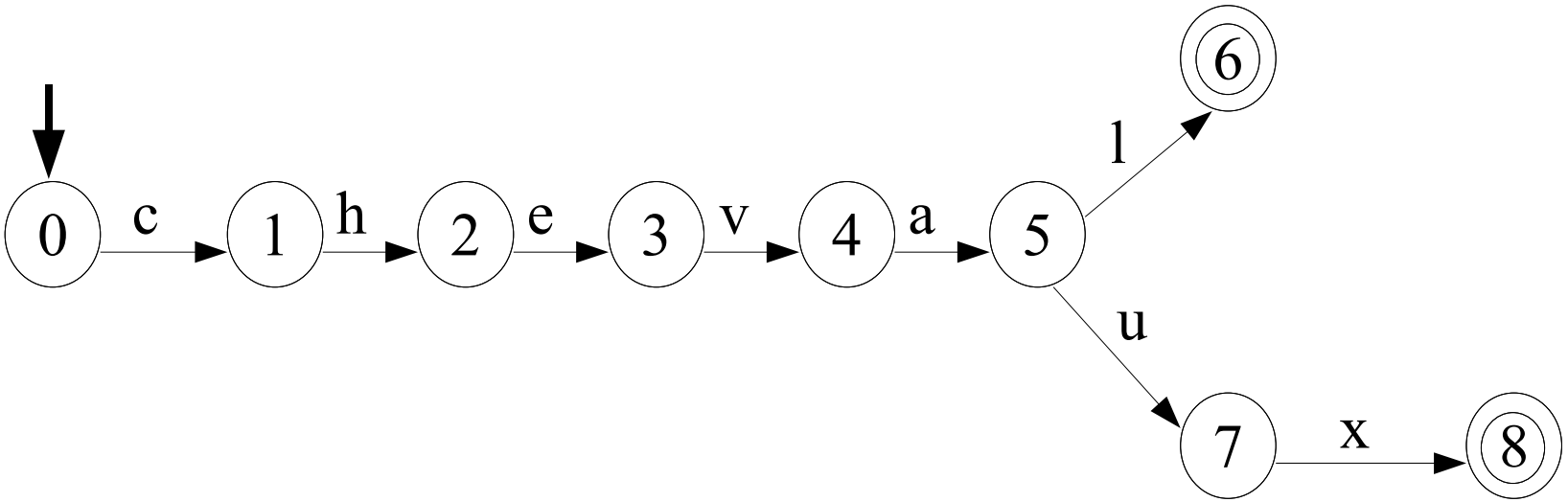
Exemple – Étape 0

```
le chevalier regarde les chevaux galoper
```



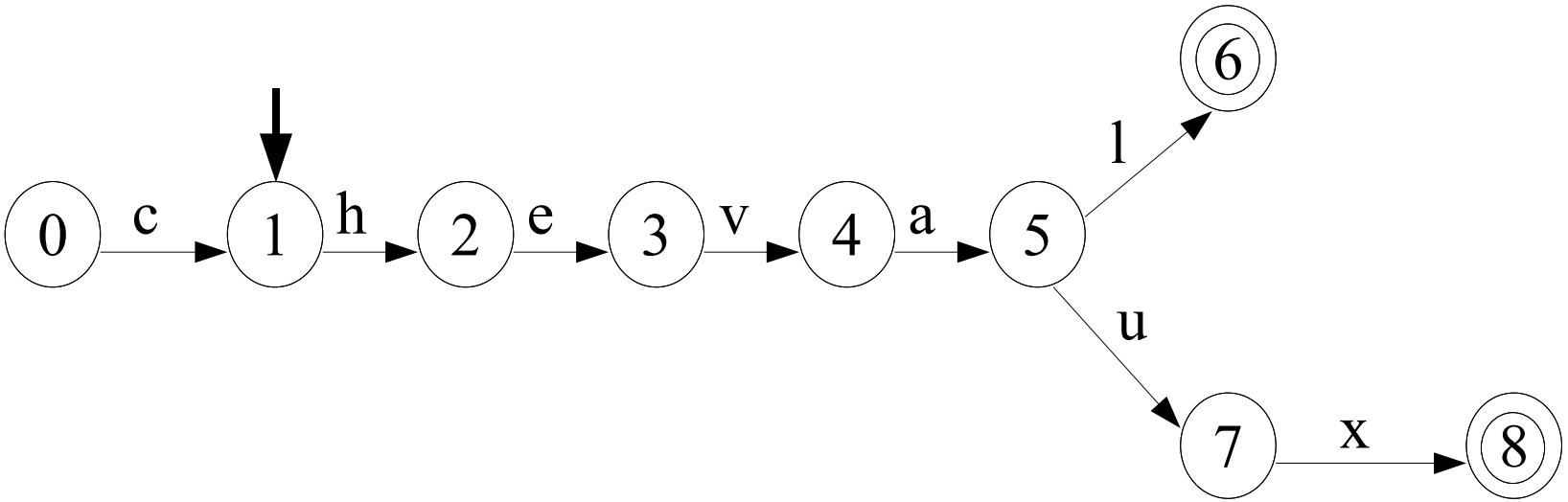
Exemple – Étape 1

le_chevalier regarde les chevaux galoper



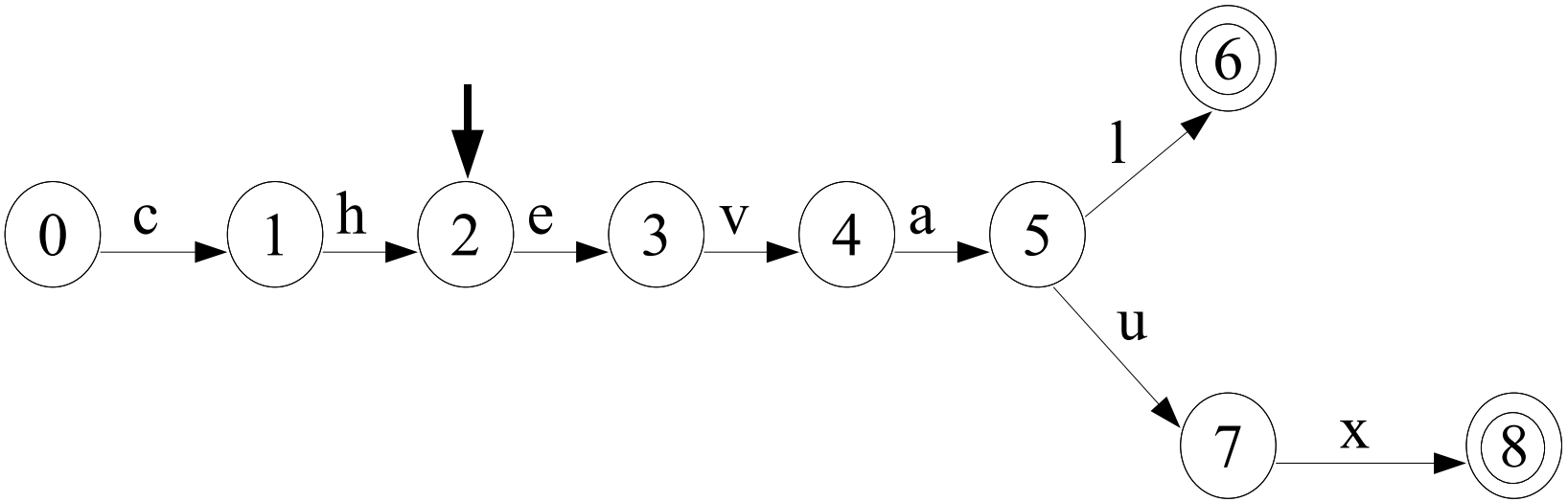
Exemple – Étape 2

le_chevalier regarde les chevaux galoper



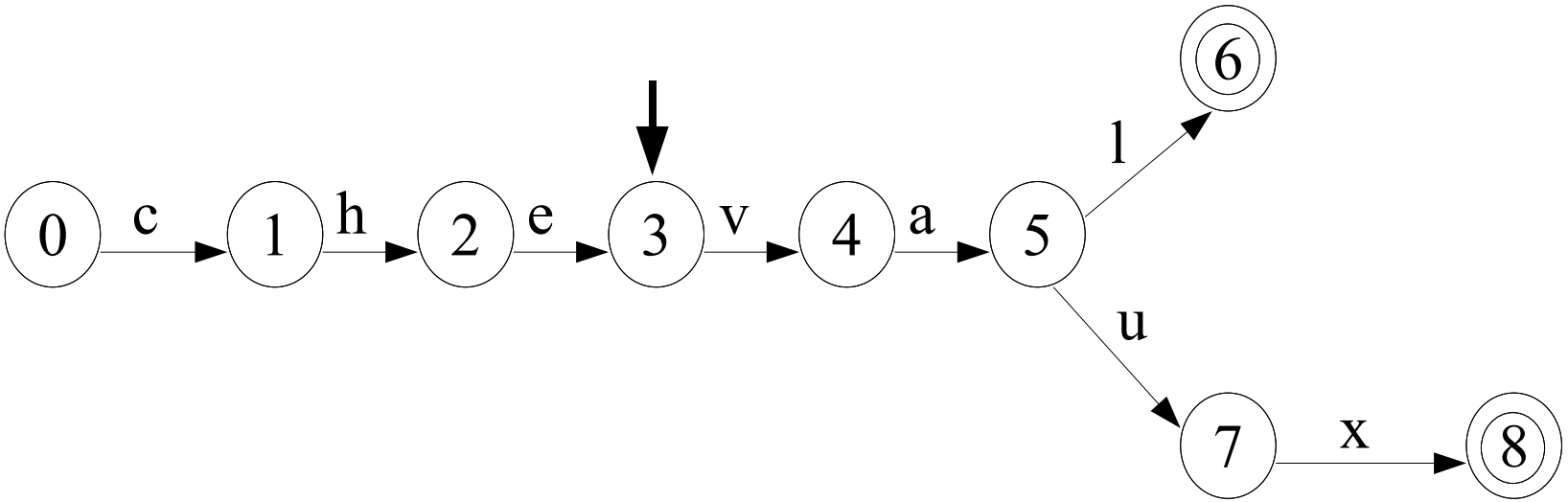
Exemple – Étape 3

le_chevalier regarde les chevaux galoper



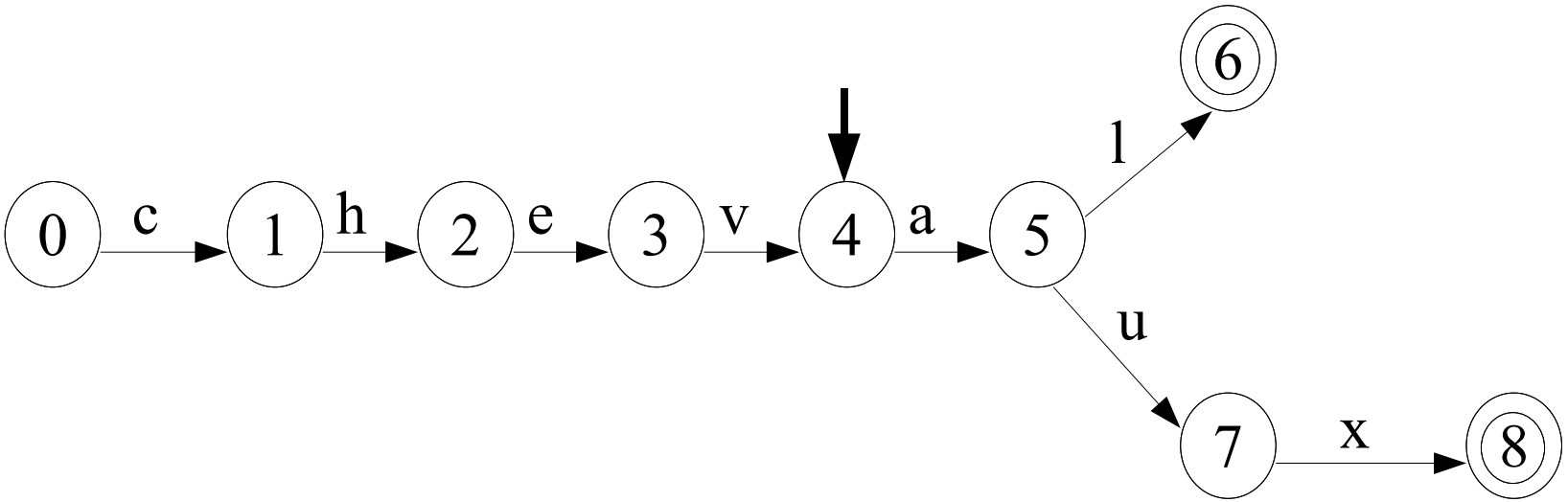
Exemple – Étape 4

le_chevalier regarde les chevaux galoper



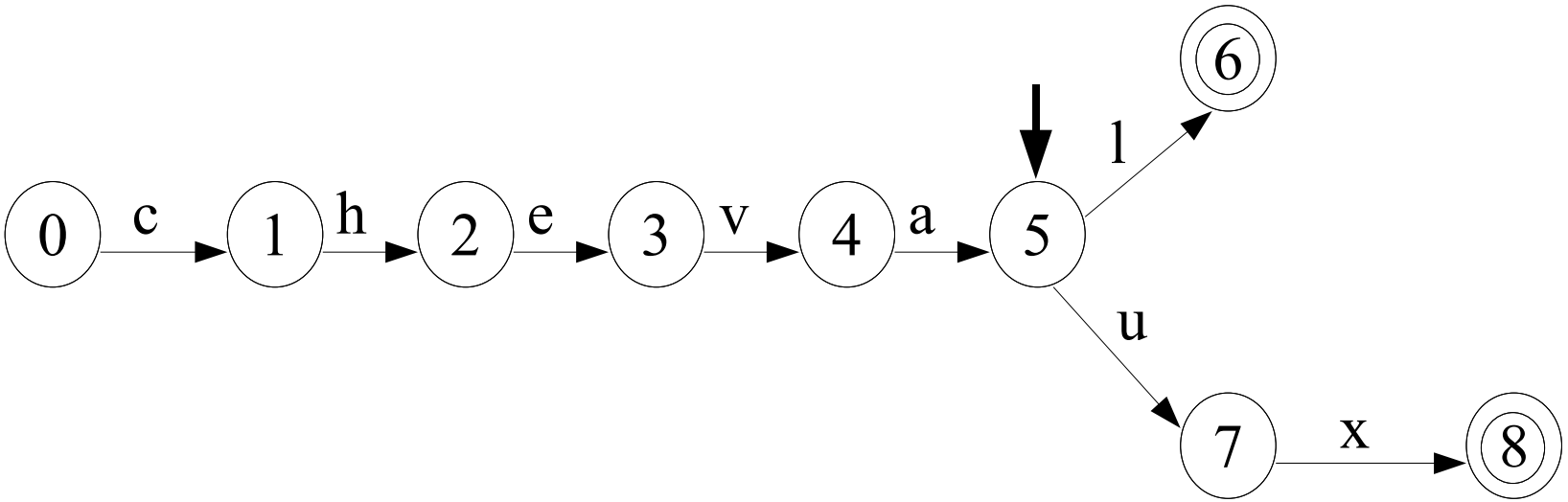
Exemple – Étape 5

le_chevalier regarde les chevaux galoper



Exemple – Étape 6

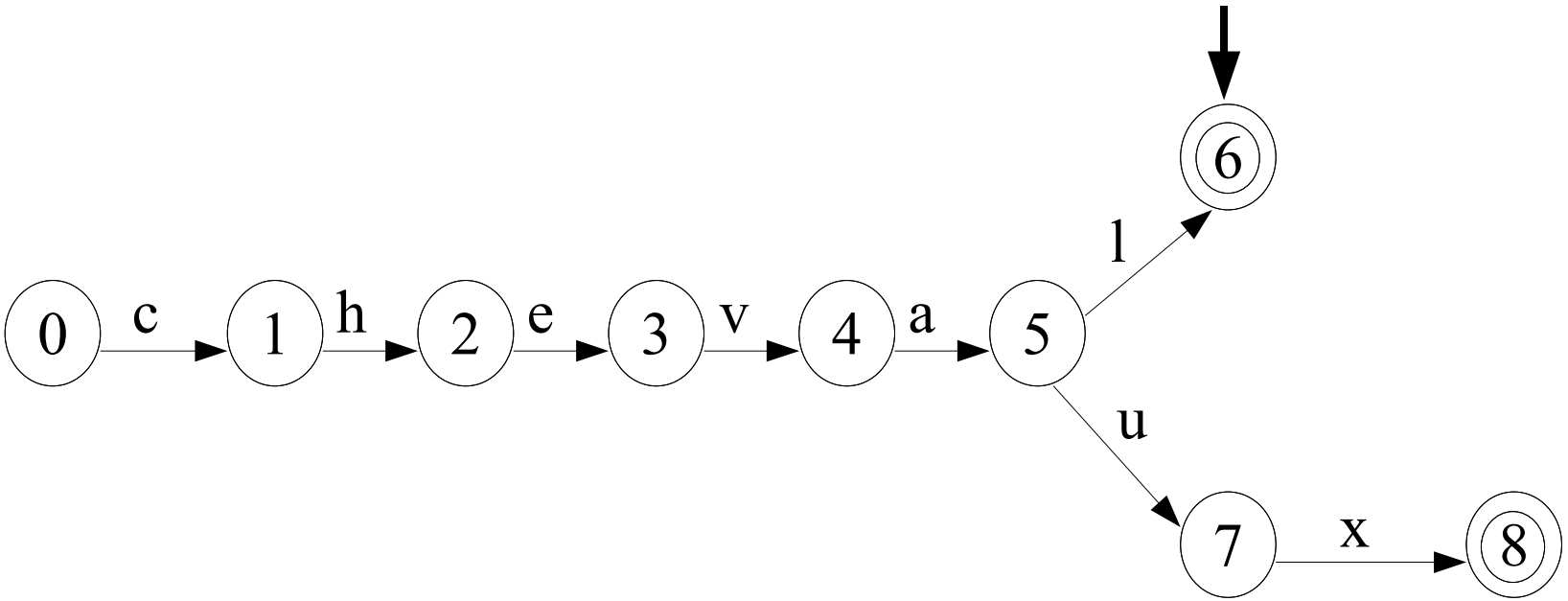
```
le_chevalier regarde les chevaux galoper
```



Exemple – Étape 7

```
le_chevalier regarde les chevaux galoper
```

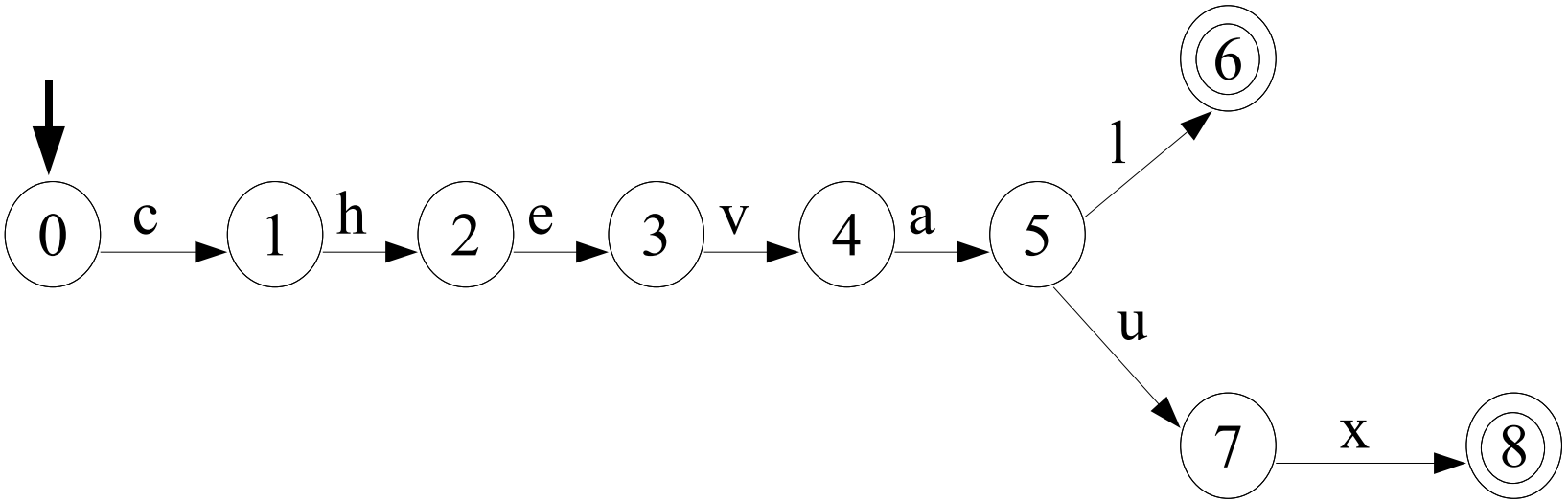
chaîne reconnue : `cheval`



Exemple – Étape 8

`le_chevalier_regarde_les_chevaux galoper`

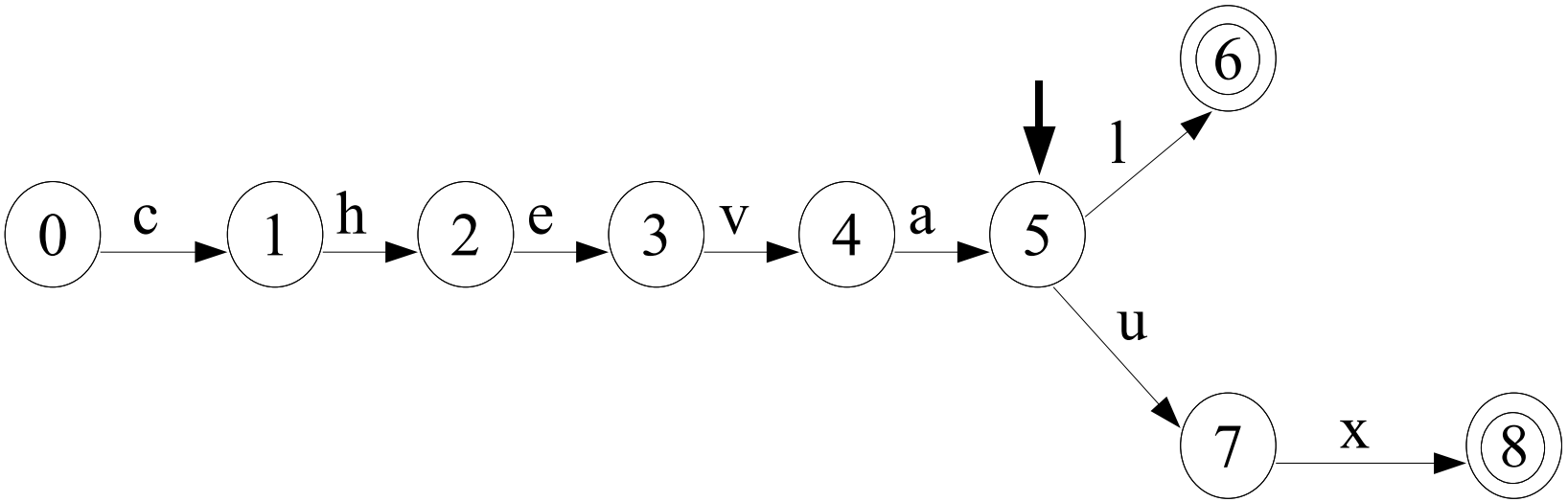
chaîne reconnue : `cheval`



Exemple – Étape 9-14

`le_chevalier_regarde_les_chevaux galoper`

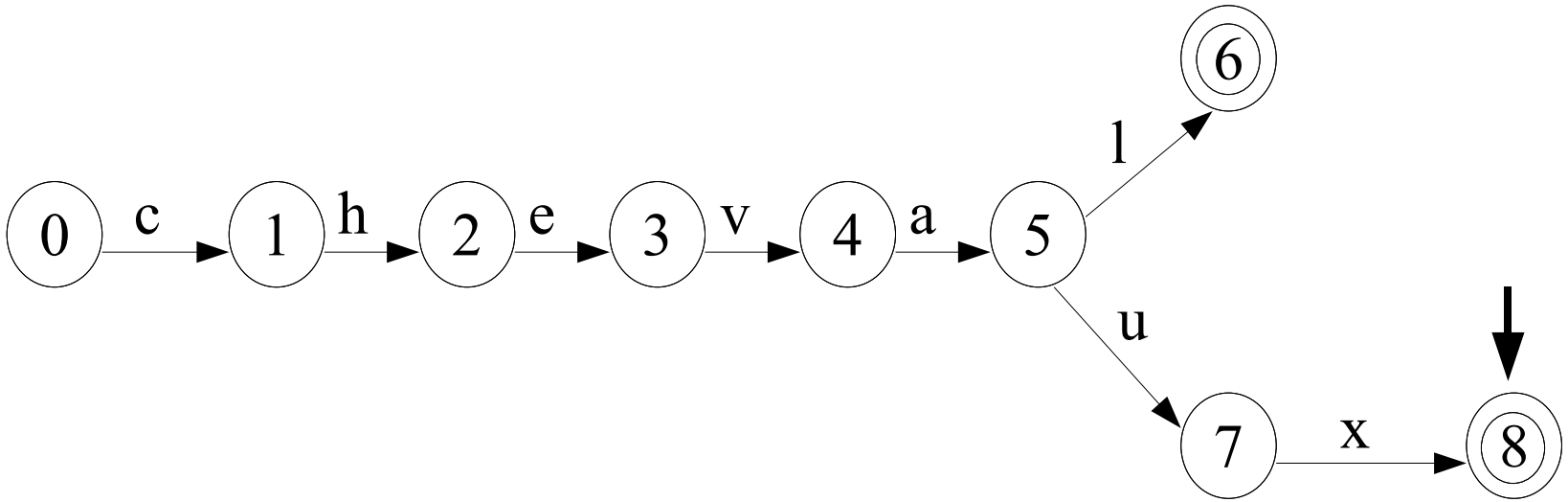
chaîne reconnue : `cheval`



Exemple – Étape 15

`le_chevalier_regarde_les_chevaux galoper`

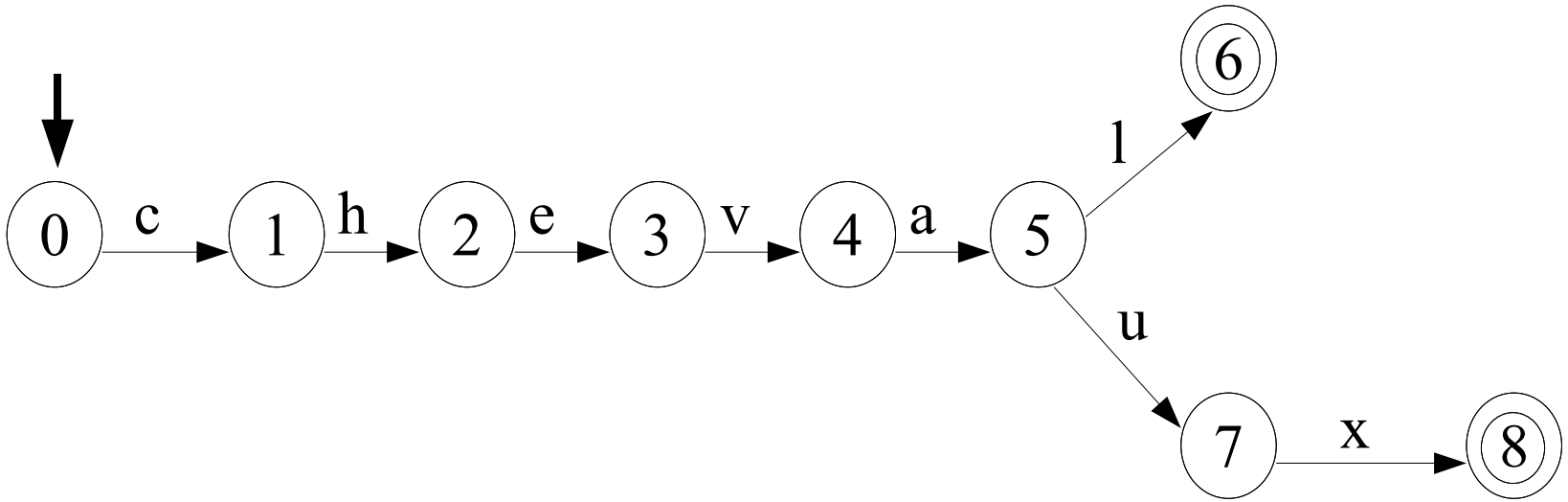
chaîne reconnue : `cheval`
chaîne reconnue : `chevaux`



Exemple – Étape 16

`le_chevalier_regarde_les_chevaux_galoper`

chaîne reconnue : `cheval`
chaîne reconnue : `chevaux`



Méta-caractères

- Nécessité d'utiliser des caractères pour définir une syntaxe des expressions régulières
- Caractères réservés :
  `( ) [ ] { } . * ? + ^ $ |`
- `\` permet de dé-spécialiser les méta-caractères
- Accès aux caractères non imprimable

\t	\n	\r
Tabulation	line feed	carriage return

Exemple simple

L'expression `a` va reconnaître la première occurrence du caractère *lettre a minuscule* dans la chaîne `Le petit chat a faim.`

Accès aux occurrences suivantes :

- avec le bouton "Rechercher suivant" dans les logiciels
- avec des options pour les outils en ligne et les langages de programmation (exemple : `g` avec `sed`)

La recherche est sensible à la casse des caractères

- en cochant l'option "ignorer la casse" dans les logiciels
- avec des options (i)

Classe de caractères

`[aeiouy]` reconnaît les voyelles

`[^aeiouy]` reconnaît tous les caractères qui ne sont pas des voyelles : `7 + é n ; '`

`[0-9]` reconnaît les chiffres

`[A-z]` reconnaît tous les caractères entre A et z donc les caractères entre Z(90) et a(97)

- les seuls méta-caractères sont : `] ^ \ -`
- bonjour

Classes prédéfinies

- \d ou [:digit:]
  tout caractère numérique : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- \w ou [:alnum:] (word metacharacter)
  une lettre [a-z] [A-Z], un chiffre [0-9]
- \s ou [:space:] (whitespace metacharacter)
  - espace, tabulation, saut de ligne, ou tout autre caractère non imprimable
- \b (word boundary metacharacter)
  - espace, ponctuation, le debut du texte, la find du texte
- Négation possible en utilisant la lettre majuscule

Caractère quelconque

. reconnaît n'importe quel caractère sauf les caractères de fin de ligne \r et \n

- Reconnaître les dates aux formats jj/mm/aa et jj:mm:aa
  - [0-9][0-9].[0-9][0-9].[0-9][0-9] reconnaît bien les formats précédents mais aussi `12345678`
  - [0-9][0-9][:/][0-9][0-9][:/][0-9][0-9]

Début/fin de ligne

- Dans les premiers outils, l'unité pour la recherche de motif était la ligne.

^A reconnaît toutes les lignes commençant par un A

\.$ reconnaît toutes les lignes terminant par un point

^$ reconnaît toutes les lignes vides

Choix

Aimer au passé simple :

- `aim(ai|as|a|âmes|âtes|èrent)`
  - reconnaît aussi `aimais`
  - est équivalent à `aim(a|âmes|âtes|èrent)`
- `␣aim(ai|as|a|âmes|âtes|èrent)␣`

Opérateur de répétition : ? * + {}

- `?` : 0 ou 1 fois
- `*` : 0 ou n fois
- `+` : 1 ou n fois
- `{}` :
  - {n} : exactement n fois
  - {n,} : au moins n fois
  - {,n} : au plus n fois
  - {n,m} : au moins n fois à au plus m fois

Exemple de répétition

- `.*` reconnaît n'importe quel séquence de caractères
- `[0-9]+` reconnaît un nombre entier
- `[0-9]+\.?[0-9]*` reconnaît un nombre entier ou décimal
- `a(bc)*` reconnaît
  - a
  - abbbbb
  - accc
  - abc
  - abcbccc

Problème de la maximisation

Par défaut les moteurs d'expressions régulières cherchent à maximiser le motif reconnu par une expression régulière

`\(.*\)` permet de reconnaître

`Le(D) chat(N) mange(V)`

`\([^)]*\)` et `\(.*?\)` permet de reconnaître

`Le(D) chat(N) mange(V)`

➔ L'opérateur `*?` permet de minimiser le motif reconnu

Remplacement : mémorisation

Il est possible de mémoriser au moyen des parenthèses les motifs reconnus et de les réutiliser ultérieurement

- Format hh:mm:ss vers hh ' mm" ss'"

  `([0-9]{2}):([0-9]{2}):([0-9]{2})`

  → $1' $2" $3'"

- (1 (2...) (3....)) (4 (5 (6....))(7 ...)))

Exemple : adresse mail

- reconnaissance utilisateur

  `[a-z0-9][a-z0-9_.-]*\@`

- reconnaissance domaine

  `[a-z0-9][a-z0-9._-]*\.`

- reconnaissance sous-domaine

  `([a-z0-9][a-z0-9._-]*\.)*`

- reconnaissance tld

  `[a-z]{2,5}`