Les grammaires formelles

max.silberztein@univ-fcome.fr

Introduction

1. Langages, grammaires et machines
2. Grammaires régulières
3. Grammaires hors-contexte
4. Grammaires contextuelles
5. Grammaires non restreintes

Références

- Beesley K., Karttunen L. Finite-state morphology: Xerox tools and techniques. CSLI, Stanford. 2003:359-75.
- Chomsky N., 1957. Syntactic Structures. Mouton & Co Eds. N.V.
- Dalrymple, M., 2001. Lexical functional grammar (Vol. 34). Brill.
- Gazdar G., Klein E., Pullum G., Sag. I., 1985. Generalized Phrase Structure Grammar. Blackwell Publishing, Oxford, England, and Harvard University Press, Cambridge, Massachusetts.
- Gross M., Lentin A. 1970. Notions sur les grammaires formelles. Gauthier-Villars Eds., Paris.
- Pollard, Carl & Ivan A. Sag. 1994. Head-Driven Phrase Structure Grammar (Studies in Contemporary Linguistics 4). Chicago, IL: The University of Chicago Press.
- Silberztein M., 2015. La formalisation des langues : l'approche de NooJ. ISTE, Eds., Londres.

https://fr.wikipedia.org/wiki/Grammaire_formelle

https://fr.wikipedia.org/wiki/Langage_formel

https://fr.wikipedia.org/wiki/Automate_fini

1. Langages, grammaires et machines

Un triple problème

- Les langues naturelles permettent de construire une infinité de phrases. Comment décrire mathématiquement un ensemble dont la taille est infinie ?

- Comment distinguer les phrases des séquences incorrectes :

  Luc mange une pomme vs. *Luc une mange pomme*

- Convergence entre :
  - Linguistes (ex. Chomsky) : décrire les langues
  - Mathématiciens (ex. Schützenberger) : décrire les ensembles infinis
  - Informaticiens (ex. Gödel) : caractériser ce qu'on peut calculer

Langages formels, définition

- Alphabet et lettres :

- Un alphabet est un ensemble fini d'éléments. Ses éléments sont les lettres de l'alphabet

- Ex. L'alphabet A_V = { a, e, i, o, u } contient 5 lettres

- Ex. L'alphabet A_F = { a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z } contient 26 lettres

Mots et langages

- A partir d'un alphabet donné, on peut construire des suites constituées de lettres : les mots

- Un langage est un ensemble de mots

- Ex. L_1 = { aime, aimes, aimons, aimez, aiment }
- Ex. L_2 = { mais, ou, et, donc, or, ni, car }
- Ex. L_3 = { dès, ès, grès, près, très }

- Comment décrire les langages qui contiennent un nombre infini de mots ?

Mot vide, langage vide, singleton du mot vide

- Le mot vide est constitué de 0 lettre
  - Les linguistes et les mathématiciens le notent ε
  - Les informaticiens le notent ""
  - Dans NooJ, on le note <E>

- Le langage vide ne contient aucun mot, ex. { } ou ∅
- Le singleton { ε } contient 1 élément : le mot vide
- Ne pas confondre mot vide et langage vide !

Les deux niveaux de langages

- Orthographe : Lettres → Alphabet → Mots → Langages :
  - les lettres, ex. « a », « ê », « ! », « . »
  - l'alphabet, ex. { a, b, c, ... z }
  - Le langage, ex. { mais, ou, et, donc, or, ni, car }

- Syntaxe : ALUs → Vocabulaire → Phrases → Langages
  - Unités linguistiques Atomiques (ALUs) = éléments du vocabulaire
  - Vocabulaire, ex. { lapin, laitue romaine, la, le, mange }
  - Ex. Langage = { « le lapin la mange », « le lapin mange la laitue romaine », « le lapin mange » }

Le niveau syntaxique : ALUs → Vocabulaire → Phrases → Langages

- Unités Linguistiques Atomiques (ALUs) = éléments du vocabulaire :

  - **Mots simples** : ALUs qui s'écrivent sous la forme de séquences de lettres entre deux séparateurs, ex. *table*
  - **Affixes** (préfixes ou suffixes), ALUs qui s'écrivent sous la forme de séquences de lettres non délimitée par deux séparateurs, ex. *remanger*, *manifestation*
  - **Mots composés** : ALUs qui s'écrivent sous la forme de séquences de lettres et de séparateurs, ex. *parce que*, *aujourd'hui*, *sous-marin*
  - **Expressions** : ALUs qui s'écrivent sous la forme de séquences de formes potentiellement discontinues, ex. *avoir ... lieu*, *prendre ... une douche*, *accuser ... le coup*, *prendre ... le taureau par les cornes*

Les deux niveaux de grammaires

- Une **grammaire orthographique** est un ensemble fini de règles qui permet d’identifier les mots d'un langage à partir de ses lettres

- Une **grammaire syntaxique** est un ensemble fini de règles qui permet d’identifier les phrases d'un langage à partir de ses ALUs

Equivalence Langages ⟺ Grammaires

- Un langage est parfaitement décrit par une grammaire

- A partir d'une grammaire orthographique, on peut produire tous les mots du langage décrit par la grammaire, et on peut vérifier qu'un mot donné appartient ou non au langage décrit

- A partir d'une grammaire syntaxique, on peut produire toutes les phrases du langage décrit par la grammaire, et on peut vérifier qu'une séquence de mots donnée appartient ou non au langage décrit

Grammaires génératives

Chomsky (linguiste américain) et Schützenberger (mathématicien français) ont inventé un mécanisme mathématique qui permet de décrire des langages : les grammaires génératives.

Grammaires génératives

- Une **grammaire générative** est un ensemble fini de **règles de réécriture**
- Chaque règle de réécriture est constituée d'un membre gauche, d'une flèche et d'un membre droit, ex. : α → β
- Exemple d'une grammaire générative syntaxique :

PHRASE → **GN** *voit* **GN**

GN → *le* **NOM**

GN → *un* **NOM**

NOM → *chat*

NOM → *chien*

Grammaires génératives

- Chaque membre gauche ou droit d'une règle de réécriture est une séquence de **symboles auxiliaires** et/ou de ***symboles terminaux*** : lettres (orthographe) ou ALU (syntaxe)

PHRASE → GN *voit* **GN**

GN → *le* **NOM**

GN → *un* **NOM**

NOM → *chat*

NOM → *chien*

Grammaires génératives

- Les noms des **symboles auxiliaires** sont arbitraires et n'ont aucune importance
- Les symboles auxiliaires peuvent être supprimés par des optimisations automatiques, sans rien changer au langage décrit
- Inutile donc de développer des théories linguistiques sur l'existence ou non de ces symboles auxiliaires...

Z37 → **X23** *voit* **X23**

X23 → *le* **T14**

GN → *un* **T14**

T14 → *chat*

T14 → *chien*

Traitement d’une grammaire par une machine : dérivation

- En partant du symbole auxiliaire initial PHRASE, on effectue n'importe quelle réécriture, c'est-à-dire qu'on remplace n'importe quel membre gauche d'une règle par son membre droit, jusqu'à épuisement des possibilités de réécriture. Par exemple :

PHRASE

PHRASE → GN *voit* GN
GN → *le* NOM
GN → *un* NOM
NOM → *chat*
NOM → *chien*

Traitement d’une grammaire par une machine : dérivation

- En partant du symbole auxiliaire initial PHRASE, on effectue n'importe quelle réécriture, c'est-à-dire qu'on remplace n'importe quel membre gauche d'une règle par son membre droit, jusqu'à épuisement des possibilités de réécriture. Par exemple :

PHRASE => **GN** *voit* **GN**

PHRASE → GN *voit* **GN**
GN → *le* **NOM**
GN → *un* **NOM**
NOM → *chat*
NOM → *chien*

Traitement d’une grammaire par une machine : dérivation

- En partant du symbole auxiliaire initial PHRASE, on effectue n'importe quelle réécriture, c'est-à-dire qu'on remplace n'importe quel membre gauche d'une règle par son membre droit, jusqu'à épuisement des possibilités de réécriture. Par exemple :

PHRASE => **GN** *voit* **GN** => *le* **NOM** *voit* **GN**

PHRASE → GN *voit* **GN**

GN → *le* **NOM**

GN → *un* **NOM**

NOM → *chat*

NOM → *chien*

Traitement d’une grammaire par une machine : dérivation

- En partant du symbole auxiliaire initial PHRASE, on effectue n'importe quelle réécriture, c'est-à-dire qu'on remplace n'importe quel membre gauche d'une règle par son membre droit, jusqu'à épuisement des possibilités de réécriture. Par exemple :

PHRASE => **GN** *voit* **GN** => *le* **NOM** *voit* **GN**

=> *le* **NOM** *voit* *un* **NOM**

PHRASE → GN *voit* **GN**

GN → *le* **NOM**

GN → *un* **NOM**

NOM → *chat*

NOM → *chien*

Traitement d’une grammaire par une machine : dérivation

- En partant du symbole auxiliaire initial PHRASE, on effectue n'importe quelle réécriture, c'est-à-dire qu'on remplace n'importe quel membre gauche d'une règle par son membre droit, jusqu'à épuisement des possibilités de réécriture. Par exemple :

PHRASE => **GN** *voit* **GN** => *le* **NOM** *voit* **GN**

=> *le* **NOM** *voit un* **NOM** => *le* **NOM** *voit un chat*

PHRASE → GN *voit* **GN**

GN → *le* NOM

GN → *un* NOM

NOM → *chat*

NOM → *chien*

Traitement d'une grammaire par une machine : dérivation

- En partant du symbole auxiliaire initial PHRASE, on effectue n'importe quelle réécriture, c'est-à-dire qu'on remplace n'importe quel membre gauche d'une règle par son membre droit, jusqu'à épuisement des possibilités de réécriture. Par exemple :

PHRASE => **GN** *voit* **GN** => *le* **NOM** *voit* **GN**

=> *le* **NOM** *voit un* **NOM** => *le* **NOM** *voit un chat*

=> *le chien voit un chat*

PHRASE → GN *voit* **GN**

GN → *le* **NOM**

GN → *un* **NOM**

NOM → *chat*

NOM → *chien*

Il n'y a plus de symbole auxiliaire ; on ne peut plus effectuer de réécriture ; la séquence *le chien voit un chat* appartient au langage décrit par la grammaire.

Exercice : grammaire, dérivation, langage

- Trouver d'autres dérivations potentielles pour la grammaire :

PHRASE → **GN** *voit* **GN**
GN → *le* **NOM**
GN → *un* **NOM**
NOM → *chat*
NOM → *chien*

- Combien de phrases cette grammaire peut-elle produite ? Quelle est la taille du langage décrite par cette grammaire ?

Exercice : grammaire, dérivation, langage

Si on calcule toutes les dérivations possibles pour la grammaire précédente, on obtient l'ensemble des 16 phrases suivantes :

{ "un chat voit un chat", "un chat voit un chien", "un chat voit le chat", "un chat voit le chien", "le chat voit un chat", "le chat voit un chien", "le chat voit le chat", "le chat voit le chien", "un chien voit un chat", "un chien voit un chien", "un chien voit le chat", "un chien voit le chien", "le chien voit un chat", "le chien voit un chien", "le chien voit le chat", "le chien voit le chien" }

PHRASE → **GN** *voit* **GN**

GN → *le* **NOM**

GN → *un* **NOM**

NOM → *chat*

NOM → *chien*

Traitement d'une grammaire par une machine : analyse

- En partant d'une séquence quelconque, on effectue n'importe quelle réécriture inverse, c'est-à-dire qu'on remplace n'importe quel membre droit d'une règle par son membre gauche, jusqu'à épuisement des possibilités de réécriture. Par exemple :

PHRASE → GN *voit* **GN**
GN → *le* **NOM**
GN → *un* **NOM**
NOM → *chat*
NOM → *chien*

le chien voit un chat => *le* **NOM** *voit un chat* =>

le **NOM** *voit un* **NOM** => *le* **NOM** *voit* **GN**=> **GN** *voit* **GN** => **PHRASE**

On a pu produire PHRASE à partir de la séquence *le chien voit un chat* ; cette séquence est donc une phrase du langage décrit par la grammaire

Traitement d'une grammaire par une machine : analyse

- En partant d'une séquence quelconque, on effectue n'importe quelle réécriture inverse, c'est-à-dire qu'on remplace n'importe quel membre droit d'une règle par son membre gauche, jusqu'à épuisement des possibilités de réécriture. Par exemple :

PHRASE → GN *voit* **GN**
GN → *le* **NOM**
GN → *un* **NOM**
NOM → *chat*
NOM → *chien*

le chien voit un cheval => *le* **NOM** *voit un cheval* =>

le **NOM** *voit un cheval* => *le* **NOM** *voit un cheval* => **GN** *voit un cheval*

On ne peut plus effectuer de réécriture inverse ; on n'a pas pu produire PHRASE à partir de la séquence *le chien voit un cheval* ; cette séquence n'est donc pas une phrase du langage décrit par la grammaire

Langages / Grammaires / Machines

- Un langage correspond à une grammaire et à une machine

- Une machine peut fonctionner dans deux sens :
  - Un **générateur** part d'un symbole auxiliaire initial (ex. PHRASE), et effectue des réécritures jusqu'à produire des phrases du langage
  - Un **analyseur** vérifie si une séquence donnée appartient à un langage en essayant d'effectuer les réécritures inverses, jusqu'à produire le symbole auxiliaire initial (ex. PHRASE)

Hiérarchie de Chomsky-Schützenberger

- Quatre types de grammaires génératives (et donc de langages, et donc de machines) :
  - Grammaires régulières (Type 3)
  - Grammaires hors contexte (Type 2)
  - Grammaires contextuelles (Type 1)
  - Grammaires non restreintes (Type 0)

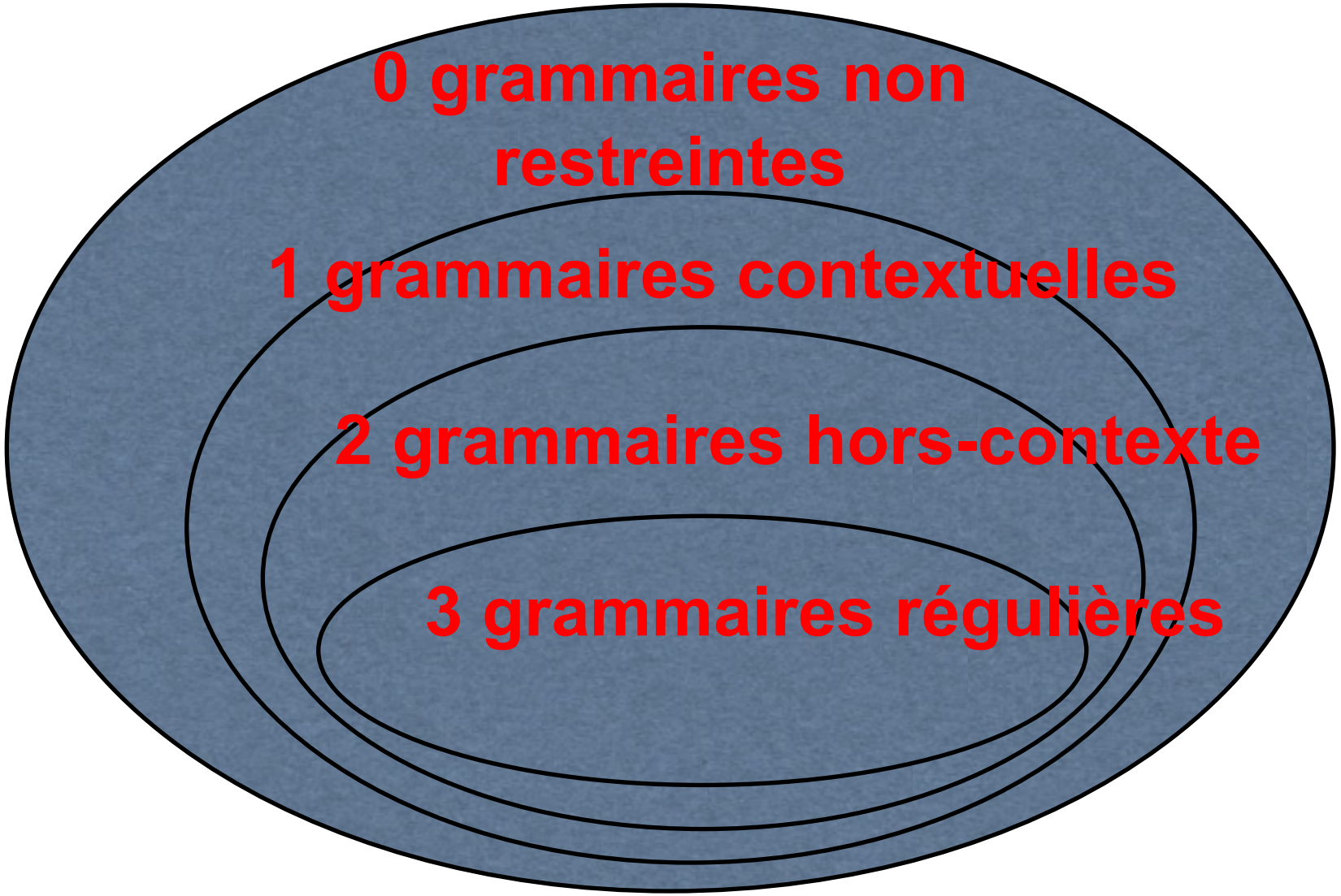
Hiérarchie de Chomsky-Schützenberger

- Quatre types de grammaires génératives, de langages, et de machines :

Grammaires régulières	Langages réguliers (ou rationnels)	Automates finis
Grammaires hors contexte	Langages algébriques	Automates à pile
Grammaires contextuelles	Langages contextuels	Automates linéairement bornés
Grammaires non restreintes	Langages récursivement énumérables	Machines de Turing

- Les grammaires hors-contexte sont plus puissantes que les grammaires régulières : elles peuvent décrire des langages que les grammaires régulières ne peuvent pas décrire.
- Les grammaires contextuelles sont plus puissantes que les grammaires hors-contexte
- les grammaires non restreintes sont plus puissantes que les grammaires contextuelles.

Hiérarchie de Chomsky-Schützenberger



Besoin de formalismes différents

- Les gramaires génératives sont beaucoup trop lourdes pour être utilisées pour décrire des langues naturelles. Par exemple, juste pour décrire la conjugaison au futur des verbes du premier groupe comme *aider* , il faudrait 6 règles + plusieurs milliers de règles **racine** :

Présent → racine *e*
Présent → racine *es*
Présent → racine *e*
Présent → racine *ons*
Présent → racine *ez*
Présent → racine *ent*

racine → *aid*
racine → *aim*
racine → *amus*
…

Besoin de formalismes adaptés

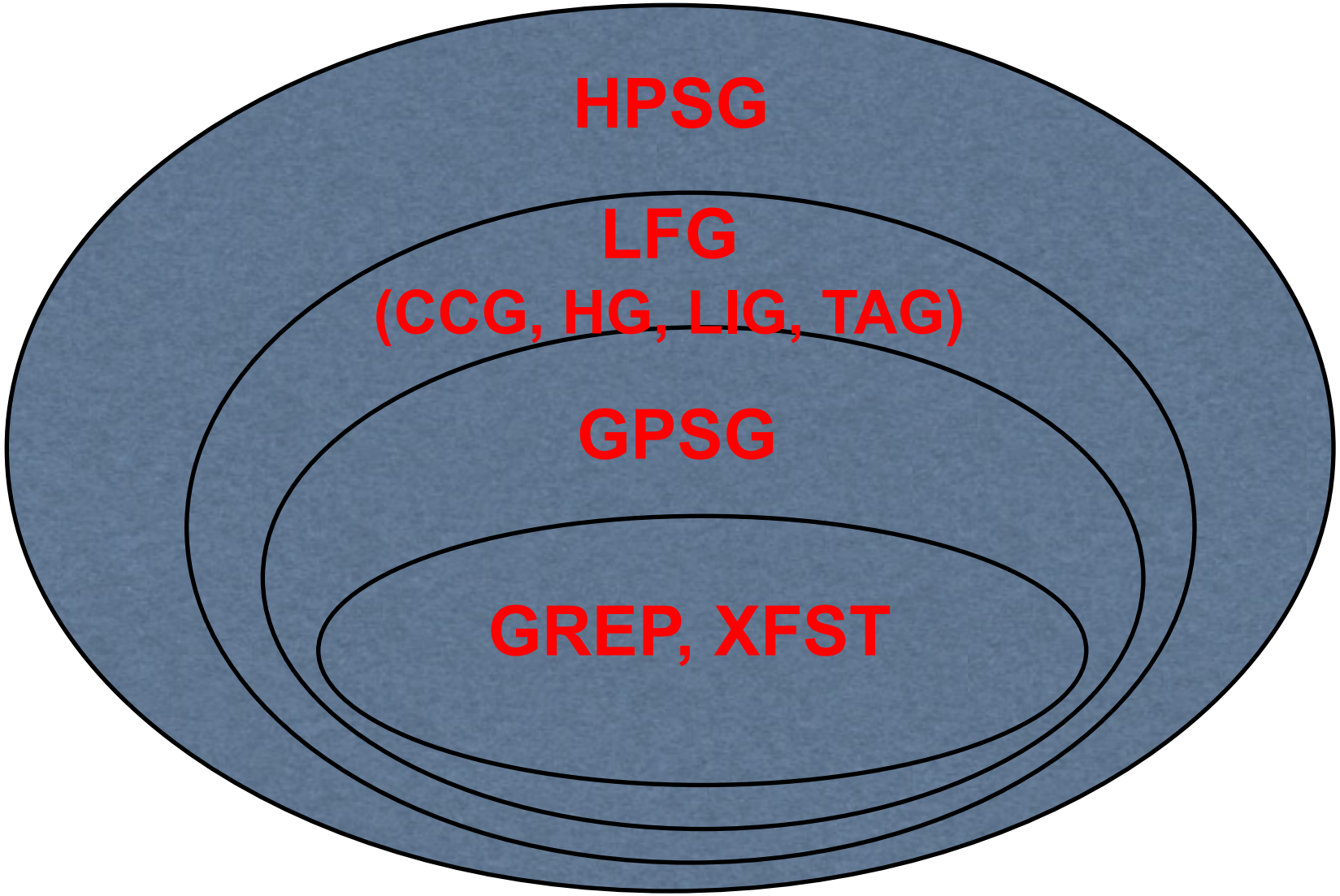
Les linguistes et linguistes-informaticiens ont donc conçus des formalismes plus adaptés à la description de gramaires.

Par exemple, les grammaires régulières utilisées en informatique :

[A-Z].*ations?

retrouver tous les fichiers dont le nom commence par une lettre majuscule non accentuée, suivie par une séquence de caractères quelconque, suivie par la séquence « ation », suivie par un « s » facultatif

Les formalismes utilisés en TAL

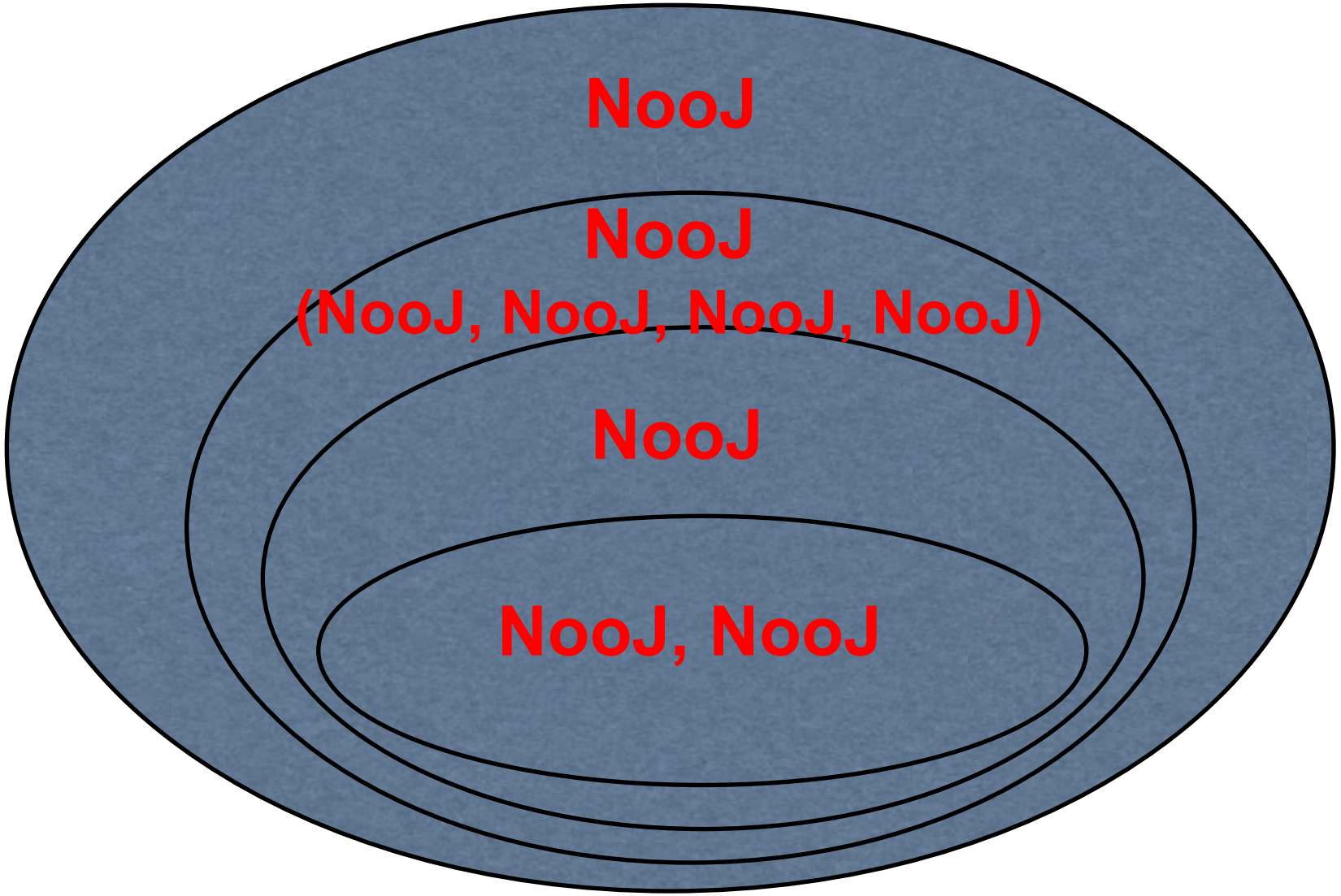


Différents formalismes : problèmes

- CCG, HG, IG, TAG : pas aussi puissants que LFG, mais les analyseurs sont plus efficaces : un compromis qui n'a pas forcément sa place en linguistique.
- Ces formalismes sont incompatibles entre eux: une règle XFST ne pourra pas être insérée dans une grammaire LFG
- Chaque grammaire doit donc être « totale », *i.e.*, doit couvrir tous les phénomènes linguistiques présents dans la phrase à représenter
- Chaque formalisme est très bien adapté à un type de phénomène linguistique... mais pas aux autres phénomènes

⇨ NooJ a été développé pour offrir les quatre types de grammaires dans un formalisme unifié.

Le formalisme de NooJ



2. Grammaires régulières

Grammaires génératives régulières (type 3)

Une grammaire générative est régulière si :

- Les membres à gauche des règles de réécriture ne contiennent qu'un et un seul symbole auxiliaire

- Les membres à droite contiennent :
  - Soit ε, ex. **GN** → ε
  - Soit une seule ALU, ex. **GN** → *Luc*
  - Soit une seule ALU suivie d'un seul symbole auxiliaire, ex. **GN** → *le* **NOM**

Grammaires génératives régulières (type 3)

- La grammaire suivante est une grammaire régulière :

PHRASE → *Jean* **GV**
GV → *voit* **GN**
GN → *un* **NOM**
NOM → *chat*
NOM → *chien*

Grammaires génératives régulières (type 3)

- La grammaire suivante est une grammaire régulière :

PHRASE → *le* **SUITE**

PHRASE → *un* **SUITE**

SUITE → *chat* **GVERBAL**

SUITE → *chien* **GVERBAL**

GVERBAL → *voit* **OBJET**

OBJET → *le* **NOM**

OBJET → *un* **NOM**

NOM → *chat*

NOM → *chien*

Grammaires génératives régulières (type 3)

- La grammaire suivante n'est pas une grammaire régulière :

PHRASE → **GN** *voit* **GN**

GN → *le* **NOM**

GN → *un* **NOM**

NOM → *chat*

NOM → *chien*

Grammaires génératives régulières (type 3)

- La grammaire suivante n'est pas une grammaire régulière :

PHRASE → **GN** *voit* **GN**

GN → *le* **NOM**

GN → *un* **NOM**

NOM → *chat*

NOM → *chien*

Grammaires régulières
Formalisme alternatif

- Une grammaire régulière est définie à partir d'un alphabet, ex. : { a, b, c }
- n'importe quelle lettre constitue une grammaire régulière, ex. : b
- le mot vide est une grammaire régulière, ex. : ε
- Si X est une grammaire régulière, alors (X) est une grammaire régulière
- Si X est une grammaire régulière, alors X* est une grammaire régulière
- Si X et Y sont deux grammaires régulières, alors X Y est une grammaire régulière
- Si X et Y sont deux grammaires régulières, alors X | Y est une grammaire régulière

Grammaires régulières

(lettres en noir, mots en bleu, grammaires en vert)

- Sur l'alphabet {a, b, c} :
- la grammaire régulière b représente le langage { b }, *i.e.*, l'ensemble qui contient le seul mot b
- Attention : ne pas confondre la lettre a avec la grammaire a ou le mot a !
- la grammaire régulière ε représente le langage { ε }, i.e., l'ensemble qui contient le mot ε
- Attention : ne pas confondre le langage { ε } avec le langage vide ∅
- la grammaire régulière (X) représente le même langage que celui représenté par la grammaire X, par ex., b = (b)

Grammaires régulières

- Sur l'alphabet {a, b, c} :
- **Concaténation** : Si X et Y sont deux grammaires régulières, alors la grammaire X Y représente l'ensemble de tous les mots qui commencent par un mot du langage reconnu par X suivi par un mot du langage reconnu par Y

Exemples :

- la grammaire X = a reconnait le langage { a }
- la grammaire Y = c reconnait le langage { c }
- la grammaire X Y reconnait le langage { ac }
- la grammaire X Y X reconnait le langage { aca }
- la grammaire X b Y c reconnait le langage { abcc }

Grammaires régulières

- Sur l'alphabet {a, b, c} :
- **Disjonction** : Si X et Y sont deux grammaires régulières, alors la grammaire X | Y représente l'ensemble de tous les mots du langage reconnu par X et aussi tous les mots du langage reconnu par Y

Exemples :

- la grammaire X = a reconnait le langage { a }
- la grammaire Y = c reconnait le langage { c }
- la grammaire X | Y reconnait le langage { a, c }
- la grammaire Y | X reconnait le langage { a, c }
- la grammaire X Y | c reconnait le langage { ac, c }

Grammaires régulières

- Sur l'alphabet {a, b, c} :
- **Opération de Kleene** : Si X est une grammaire régulière, alors la grammaire X* représente l'ensemble de tous les mots que l'on peut contruire avec des mots du langage reconnu par X

Exemples :

- la grammaire a reconnait le langage { a }
- la grammaire a* reconnait le langage { ε, a , aa , aaa, aaaa, aaaaa, ... }
- la grammaire (aba)* reconnait le langage { ε, aba , abaaba, abaabaaba, ... }
- la grammaire (a|b)* reconnait le langage { ε, a , b, ab, aa, bb, ba, aaaba, ... }

Utilité des parenthèses

(priorité de l'opérateur de concaténation par rapport à l'opérateur de disjonction)

- En arithmétique, on a :

$$2 \times 3 + 4 = 6 + 4 = 10$$

$$\neq$$

$$2 \times (3 + 4) = 2 \times 7 = 14$$

- En algèbre, on a :

$$2\ (a + b) = 2\ a + 2\ b \neq 2\ a + b$$

- En linguistique, on a de même :

la (chaise | table) = la chaise | la table

≠

la chaise | table

Exercice

Sur l'alphabet { a, b, c }, construire
la grammaire qui représente le langage qui contient :

- tous les mots de quatre lettres

Exercice

Sur l'alphabet { a, b, c }, construire
la grammaire qui représente le langage qui contient :

- tous les mots de quatre lettres

(a|b|c) (a|b|c) (a|b|c) (a|b|c)

Exercice

Sur l'alphabet { a, b, c }, construire
la grammaire qui représente le langage qui contient :

- tous les mots qui se terminent par la lettre « c »

Exercice

Sur l'alphabet { a, b, c }, construire
la grammaire qui représente le langage qui contient :

- tous les mots qui se terminent par la lettre « c » :

(a|b|c)* c

Exercice

Sur l'alphabet { a, b, c }, construire
la grammaire qui représente le langage qui contient :

- tous les mots qui ne contiennent pas la lettre « b »

Exercice

Sur l'alphabet { a, b, c }, construire
la grammaire qui représente le langage qui contient :

- tous les mots qui ne contiennent pas la lettre « b »

(a|c)*

Exercice

Sur l'alphabet { a, b, c }, construire
la grammaire qui représente le langage qui contient :

- tous les mots qui contiennent une lettre « b » et une seule

Exercice

Sur l'alphabet { a, b, c }, construire
la grammaire qui représente le langage qui contient :

- tous les mots qui contiennent une lettre « b » et une seule

(a|c)* b (a|c)*

Exercice

Sur l'alphabet { a, b, c }, construire
la grammaire qui représente le langage qui contient :

- tous les mots qui contiennent le mot « bac »

Exercice

Sur l'alphabet { a, b, c }, construire
la grammaire qui représente le langage qui contient :

- tous les mots qui contiennent le mot « bac »

(a|b|c)* bac (a|b|c)*

Exercice

Sur l'alphabet { a, b, c }, construire

la grammaire qui représente le langage qui contient :

- tous les mots qui commencent par un « a » et se terminent par un « b »

Exercice

Sur l'alphabet { a, b, c }, construire

la grammaire qui représente le langage qui contient :

- tous les mots qui commencent par un « a » et se terminent par un « b »

a (a|b|c)* b

Exercices à rendre

Sur l'alphabet { a, b, c }, construire la grammaire

qui représente le langage qui contient :

1. (1 pt) dont la longueur est un multiple de 2
2. (1 pt) qui ne contiennent que des « a », ou alors que des « b »
3. (1 pt) qui ont un nombre pair de « a »

Sur l'alphabet français, construire la grammaire

qui représente le langage qui contient :

4. (1 pt) tous les mots communs français qui contiennent un « ë »
5. (1 pt) toutes les formes conjuguées du verbe *voler*
6. (2 points) Sur le vocabulaire V = {il, donne, le, la, les, lui, leur, me, te, se, nous, vous, en, y}, construire la grammaire qui représente le langage qui contient toutes les séquences de mots correctes en français

Grammaires régulières augmentées

- Le but n'est pas seulement de reconnaître des séquences : on veut aussi produire des résultats. Le résultat peut être une analyse linguistique, une traduction, une représentation sémantique, etc.
- On utilise deux alphabets : l'alphabet de lecture (input) et l'alphabet d'écriture (output)
- La grammaire met en relation les séquences lues (input) avec les séquences écrites (output)
- Exemples :
  - Remplacer dans un texte toutes les occurrences de « a » par des « b » : a/b
  - Mettre toutes les occurrences de « monday » en majuscule : monday/Monday
  - Traduire l'expression « quelle heure est-il ? » en anglais :

    quelle/what heure/time est/is-il/it ?/?

Exercice

- Associer toutes les formes du nom *cousin* aux propriétés +m, +f, +s et +p

Exercice

- Associer toutes les formes du nom *cousin* aux propriétés +m, +f, +s et +p

c o u s i n (e/+f+s | s/+m+p | es/+f+p | ɛ/+m+s)

Exercice

- Associer toutes les formes conjuguées du verbe *voler* au présent aux propriétés +1, +2, +3, +s et +p

Exercice

- Associer toutes les formes conjuguées du verbe *voler* au présent aux propriétés +1, +2, +3, +s et +p

vol (e/+1+s | es/+2+s | e/+3+s | ons/+1+p | ez/+2+p | ent/+3+p)

Exercice

- Associer toutes les formes conjuguées du verbe *voler* à l'imparfait aux propriétés +1, +2, +3, +s et +p

Exercice

- Associer toutes les formes conjuguées du verbe *voler* à l'imparfait et au conditionnel présent aux propriétés +1, +2, +3, +s et +p

Exercice

- Associer toutes les formes conjuguées du verbe *voler* à l'imparfait aux propriétés +1, +2, +3, +s et +p

vol (ais/+1+s | ais/+2+s | ait/+3+s | ions/+1+p | iez/+2+p | aient/+3+p)

Exercice

- Associer toutes les formes conjuguées du verbe *voler* à l'imparfait et au conditionnel présent aux propriétés +1, +2, +3, +s et +p

vol (ɛ|er) (ais/+1+s | ais/+2+s | ait/+3+s | ions/+1+p | iez/+2+p | aient/+3+p)

Exercice

- Associer toutes les variantes orthographiques et morphologiques du nom *tsar* aux propriétés +m, +f, +s et +p

Exercice

- Associer toutes les variantes orthographiques et morphologiques du nom *tsar* aux propriétés +m, +f, +s et +p

(c|t) (s|z) ar (ε/+m | ine/+f) (ε/+s | s/+p)

Grammaire régulières dans NooJ

- Notation dans NooJ (le mot vide ε s'écrit ici <E>)

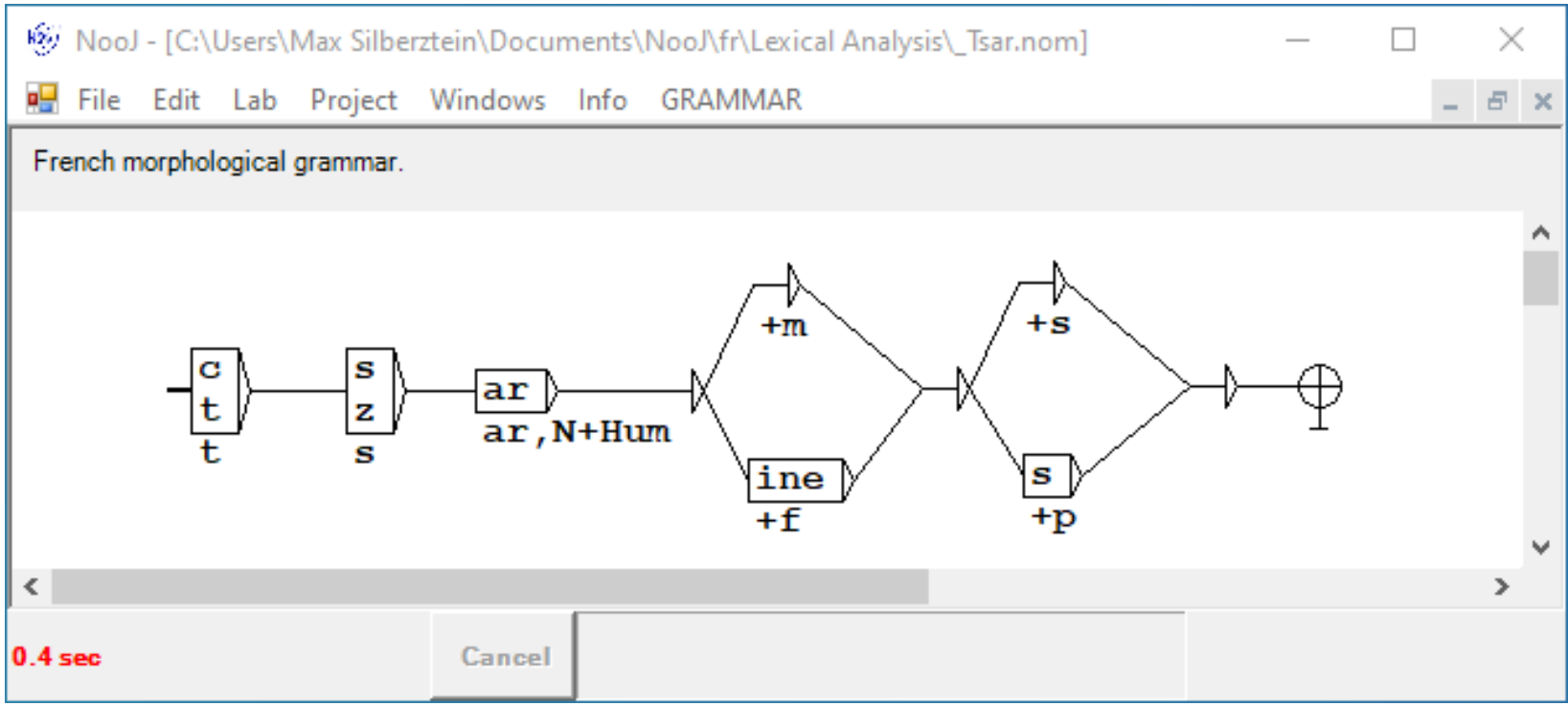
(c|t) (s|z) ar (<E>/+m | ine/+f) (<E>/+s | s/+p)

- Notation graphique équivalente dans NooJ :

Ce graphe reconnait et annote les 16 formes du mot *tsar.* Par ex., la forme *tsarines* sera annotée :

tsar,N+Hum+f+p

(*tsarines* est un forme du mot *tsar*, c'est un nom humain féminin pluriel).



Exercice

- Reconnaître toutes les variantes orthographiques du terme *Moyen-Âge*, et les normaliser

Exercice

- Reconnaître toutes les variantes orthographiques du terme *Moyen-Âge* et les normaliser

(M | m)/M oyen (— | ɛ)/— (Â|A|â)/Â ge |

MOYEN/Moyen (— | ɛ)/— (Â|A)/Â GE/ge

Exercice

- Reconnaître toutes les variantes orthographiques du terme *Moyen-Âge* et les normaliser

(M | m)/M oyen (— | ɛ)/— (Â|A|â)/Â ge |
MOYEN/Moyen (— | ɛ)/— (Â|A)/Â GE/ge

=

((M | m) oyen (— | ɛ) (Â|A|â) ge |
MOYEN (— | ɛ) (Â|A) GE/ge)/Moyen-Âge

Exercice

- Lever l’ambiguïté des mots *le*, *la* et les lorsqu’ils apparaissent entre un pronom personnel et un verbe, ex. *la mange*, *les donne*, *la donne*, etc.

Exercice

- Lever l'ambiguïté des mots *le*, *la* et *les* lorsqu'ils apparaissent entre un pronom personnel et un verbe, ex. *il la donne*, *elle ne les donne pas*, *ils le leur donne*, etc.

(le|la|les)/PRONOM (ɛ | lui | leur) VERBE

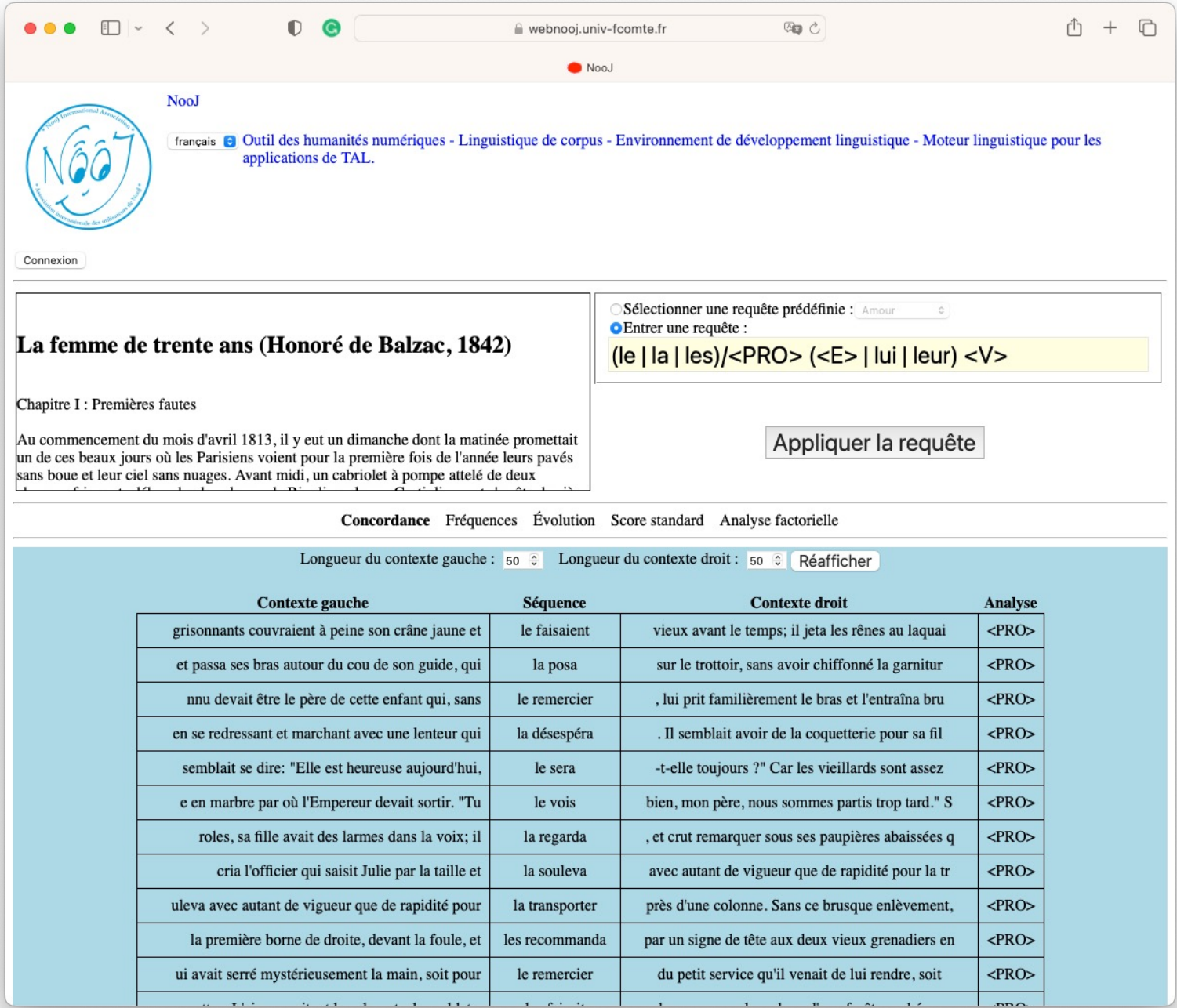
Exemple de grammaire régulière NooJ

Notation NooJ :

- utilise la notation simplifiée

- le mot vide ε est ici noté <E>

- les catégories sont écrites entre crochets, ex. <N> = nom, <V> = verbe, <A> = adjectif, <PRO> = pronom, etc.



Contexte gauche	Séquence	Contexte droit	Analyse
grisonnants couvraient à peine son crâne jaune et	le faisaient	vieux avant le temps; il jeta les rênes au laquai	<PRO>
et passa ses bras autour du cou de son guide, qui	la posa	sur le trottoir, sans avoir chiffonné la garnitur	<PRO>
nnu devait être le père de cette enfant qui, sans	le remercier	, lui prit familièrement le bras et l'entraîna bru	<PRO>
en se redressant et marchant avec une lenteur qui	la désespéra	. Il semblait avoir de la coquetterie pour sa fil	<PRO>
semblait se dire: "Elle est heureuse aujourd'hui,	le sera	-t-elle toujours ?" Car les vieillards sont assez	<PRO>
e en marbre par où l'Empereur devait sortir. "Tu	le vois	bien, mon père, nous sommes partis trop tard." S	<PRO>
roles, sa fille avait des larmes dans la voix; il	la regarda	, et crut remarquer sous ses paupières abaissées q	<PRO>
cria l'officier qui saisit Julie par la taille et	la souleva	avec autant de vigueur que de rapidité pour la tr	<PRO>
uleva avec autant de vigueur que de rapidité pour	la transporter	près d'une colonne. Sans ce brusque enlèvement,	<PRO>
la première borne de droite, devant la foule, et	les recommanda	par un signe de tête aux deux vieux grenadiers en	<PRO>
ui avait serré mystérieusement la main, soit pour	le remercier	du petit service qu'il venait de lui rendre, soit	<PRO>

Exercice

- Construire une grammaire pour lever l'ambiguïté des mots *par* et *pour* (préposition ou nom ?)

Exercice

- Construire une grammaire pour lever l'ambiguïté des mots *par* et *pour* (préposition ou nom ?)

pour/PREPOSITION (ε|ne) (ε|jamais|pas|plus) (ε|me) (ε|le) (ε|leur|lui) VERBE+Infinitif |
(par|pour)/PREPOSITION DETERMINANT |

le/DETERMINANT pour/NOM |

DETERMINANT+m+s par/NOM

Levée d’ambiguïté morpho-syntaxique automatique

- Des dizaines de grammaires de levée d’ambiguïté peuvent être développées autour de mots fréquents comme :
    - *est* : verbe, nom ou adjectif ?
    - *si* : conjonction ou nom ?
    - *place* : nom ou verbe ?
    - *fait* : nom ou verbe ?
    - *a* : nom ou verbe ?
    - *son, ton* : déterminants ou noms ?
    - *plus* : verbe ou adverbe ?
    - *rue* : verbe ou nom ?
    - *complet* : adjectif ou nom ?
    - première : adjectif ou nom ?
    - …

3. Grammaires hors-contexte

Grammaires génératives hors contexte (type 2)

Une grammaire générative est hors-contexte si :

- Les membres à gauche des règles de réécriture ne contiennent qu'un et un seul symbole auxiliaire

- Les membres à droite contiennent n'importe quelle séquence d'ALU, de symboles et/ou de mots vides, ex. : **GN → DET NOM *de* NOM**

Grammaires génératives hors contexte (type 2)

- La grammaire suivante est une grammaire hors contexte :

PHRASE → **GN** *voit* **GN**

GN → *le* **NOM**

GN → *un* **NOM**

NOM → *chat*

NOM → *chien*

Grammaires génératives hors contexte (type 2)

- La grammaire suivante n'est pas une grammaire hors contexte :

PHRASE → GN *voit* **GN**
GN Singulier → *le* **NOM**
GN Pluriel → *les* **NOM**
NOM → *chat*
NOM → *chats*

Grammaires génératives hors contexte (type 2)

- La grammaire suivante n'est pas une grammaire hors contexte :

PHRASE → **GN** *voit* **GN**
GN Singulier → *le* **NOM**
GN Pluriel → *les* **NOM**
NOM → *chat*
NOM → *chats*

Les grammaires hors contexte, formalismes alternatifs
(ici : notation Backus-Naur)

- Le membre droit des règles de réécriture peut contenir des disjonctions « | »:

<unités> ::= I | II | III | IV | V | VI | VII | VIII | IX

<dizaines> ::= X | XX | XXX | XL | L | LX | LXX | LXXX | XC

- On peut réutiliser les expressions nommées dans des règles :

<chiffre-romain> ::= <dizaines> <unités> | <dizaines> ""

- Une grammaire hors-contexte contient donc un ensemble de règles nommées

Les grammaires hors contexte, formalismes alternatifs
(ici : notation simplifiée)

- Le membre droit peut contenir des expressions avec disjonctions « | », des parenthèses, des opérateurs de Kleene « * » et des mots vides ε :

Unités = I | II | III | IV | V | VI | VII | VIII | IX

Dizaines = X | XX | XXX | XL | L | LX | LXX | LXXX | XC

ChiffreRomain = Dizaines (Unités | ε)

Exercice

Compléter la grammaire des chiffres romains :

Unités = I | II | III | IV | V | VI | VII | VIII | IX

Dizaines = X | XX | XXX | XL | L | LX | LXX | LXXX | XC

Centaines = ...

Milliers = ...

ChiffreRomain = ...

Exercice

- Compléter la grammaire des chiffres romains :

Unités = I | II | III | IV | V | VI | VII | VIII | IX

Dizaines = X | XX | XXX | XL | L | LX | LXX | LXXX | XC

Centaines = C | CC | CCC | CD | D | DC | DCC | DCCC | CM

Milliers = M | MM | MMM

ChiffreRomain = (ε|Milliers) (ε|Centaines) (ε|Dizaines) (ε|Unités)

Première tentative : les chiffres des milliers, des centaines, des dizaines et des unités peuvent être absents, ex. « MMMII »

Exercice

- Compléter la grammaire des chiffres romains :

Unités = I | II | III | IV | V | VI | VII | VIII | IX

Dizaines = X | XX | XXX | XL | L | LX | LXX | LXXX | XC

Centaines = C | CC | CCC | CD | D | DC | DCC | DCCC | CM

Milliers = M | MM | MMM

ChiffreRomain = ~~(ε|Milliers) (ε|Centaines) (ε|Dizaines) (ε|Unités)~~

PROBLÈME : cette grammaire reconnaît le mot vide :(

Exercice

- Compléter la grammaire des chiffres romains :

Unités = I | II | III | IV | V | VI | VII | VIII | IX

Dizaines = X | XX | XXX | XL | L | LX | LXX | LXXX | XC

Centaines = C | CC | CCC | CD | D | DC | DCC | DCCC | CM

Milliers = M | MM | MMM

ChiffreRomain = (ε|Milliers) (ε|Centaines) (ε|Dizaines) Unités |
(ε|Milliers) (ε|Centaines) Dizaines |
(ε|Milliers) Centaines |
Milliers

Exercice : Grammaire hors-contexte augmentée

- En plus de reconnaître les chiffres romains, produire leur valeur en chiffres arabes.

Unités = I | II | III | IV | V | VI | VII | VIII | IX

Dizaines = X | XX | XXX | XL | L | LX | LXX | LXXX | XC

Centaines = C | CC | CCC | CD | D | DC | DCC | DCCC | CM

Milliers = M | MM | MMM

ChiffreRomain = (ε|Milliers) (ε|Centaines) (ε|Dizaines) Unités |

(ε|Milliers) (ε|Centaines) Dizaines |

(ε|Milliers) Centaines |

Milliers

Exercice : Grammaires hors-contexte augmentées

- En plus de reconnaître les chiffres romains, produire leur valeur en chiffres arabes.

Unités = I/1 | II/2 | III/3 | IV/4 | V/5 | VI/6 | VII/7 | VIII/8 | IX/9

Dizaines = X/1 | XX/2 | XXX/3 | XL/4 | L/5 | LX/6 | LXX/7 | LXXX/8 | XC/9

Centaines = C/1 | CC/2 | CCC/3 | CD/4 | D/5 | DC/6 | DCC/7 | DCCC/8 | CM/9

Milliers = M/1 | MM/2 | MMM/3

ChiffreRomain = ε/VAL=

Milliers (ε/0|Centaines) (ε/0|Dizaines) (ε/0|Unités) |

Centaines (ε/0|Dizaines) (ε/0|Unités) |

Dizaines (ε/0|Unités) |

Unités

Grammaires hors-contexte, notation de NooJ

Notation NooJ :

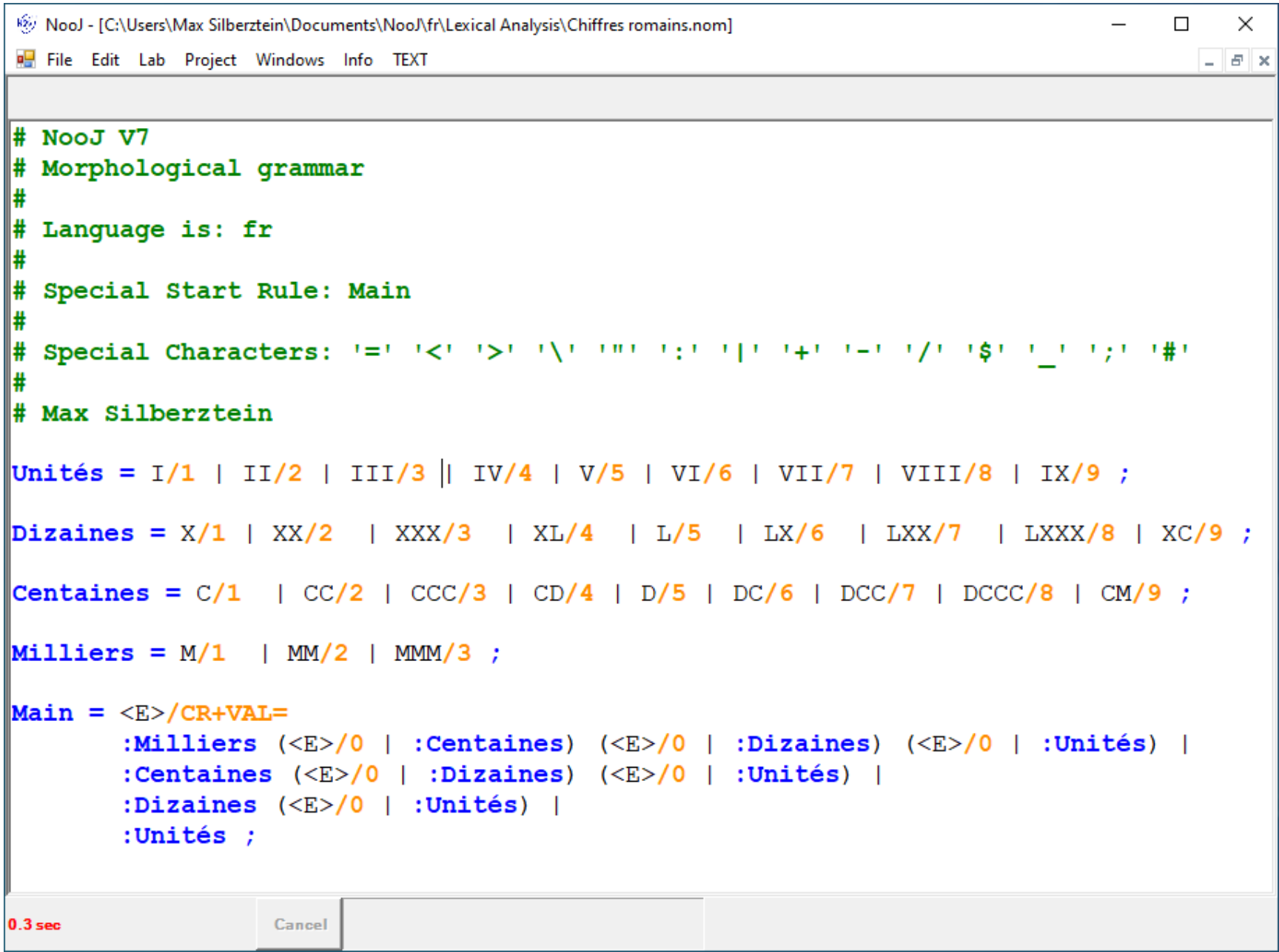
- utilise la notation simplifiée

- le mot vide ε est ici noté <E>

- les symboles auxiliaires sont préfixés par le caractère « : »

- chaque règle doit être terminée par un point virgule « ; »

- la règle principale doit s'appeler **Main**



Exercice

Construire une grammaire des groupes nominaux féminins singuliers que l'on peut construire avec des déterminants, des noms, des adjectifs.

GN+f+s =

Exercice

Construire une grammaire des groupes nominaux féminins singuliers que l'on peut construire avec des déterminants, des noms, des adjectifs.

GN+f+s = DET (ε | ModifG) NOM (ε | ModifD)

ModifG = très* ADJ

ModifD = ADJ* (ε | de (GN | NOM))

ADJ = belle | grande | jolie | petite | ...

DET = cette | la | ma | ta | sa | notre | votre | leur | une | ...

NOM = chaise | fleur | maison | table | ...

Récursivité

- Dans une grammaire hors-contexte, l'expression (à droite) peut contenir des symboles auxiliaires

- Que se passe-t-il lorsqu'une règle est définie à partir d'elle-même, ex. :

Phrase = **Sujet Verbe** (ε | *que* **Phrase**)
Sujet = *Jean* | *Marie* | *Paul* | *Hélène*
Verbe = *dit* | *espère* | *pense* | *dort*

Récursivité

- Dans une grammaire hors-contexte, l'expression (à gauche) peut contenir des symboles auxiliaires

- Que se passe-t-il lorsqu'une règle est définie à partir d'elle-même, ex. :

Phrase = **Sujet Verbe** (ε | *que* **Phrase**)
Sujet = *Jean* | *Marie* | *Paul* | *Hélène*
Verbe = *dit* | *espère* | *pense* | *dort*

Jean dit que Marie espère que Paul pense que Hélène dort

Trois sortes de récursivité

- Récursivité à droite

  Phrase = **GN** (*pense* | *dit* | *espère* | *voit*) **GN**

  GN = *son voisin* | *que* **Phrase**

- Récursivité à gauche

  GN = **GN** (*rouge* | *de Paul* | *de campagne* | *en Bretagne*) | *la maison*

- Récursivité générale

  Phrase = **DET NOM** que **Phrase VERBE**

Supprimer la récursivité à gauche

- Récursivité à gauche :

  GN = **GN** (*rouge* | *de Paul* | *de campagne*| *en Bretagne*) | *la maison*

- On peut supprimer les récursivités à gauche et construire la grammaire régulière équivalente :

  GN = *la maison* (*rouge* | *de Paul* | *de campagne* | *en Bretagne*)*

Supprimer la récursivité à droite

- Récursivité à droite :

  Phrase = **GN** (*pense* | *dit* | *espère* | *voit*) **GN**

  GN = *son voisin* | *que* **Phrase**

- On peut supprimer les récursivités à droite et construire la grammaire régulière équivalente :

  Phrase = (*son voisin* (*pense* | *dit* | *espère* | *voit*) *que*)*

  son voisin (*pense* | *dit* | *espère* | *voit*) *son voisin*

Supprimer la récursivité générale

- Récursivité générale (ni à gauche, ni à droite) :

  Phrase = DET NOM (ε | Relative) Verbe

  Relative = *que* Phrase

  Verbe = *dort* | *aime* | *a vu* | *a embauchée*

- Impossible de supprimer cette récursivité…

Supprimer la récursivité générale

- Récursivité générale :

  Phrase = DET NOM (ε | Relative) Verbe

  Relative = *que* Phrase

  Verbe = *dort* | *aime* | *a vu* | *a embauchée*

- Mais y a-t-il vraiment un besoin linguistique ?

Le chat dort. Le chat que la cousine a vu dort

? Le chat que la voisine que le voisin aime a vu dort

** Le chat que la voisine que le voisin que la police recherche aime a vu dort*

Supprimer la récursivité générale

- Récursivité générale :

  Phrase = <DET> <N> (ε | Relative) Verbe

  Relative = *que* Phrase

  Verbe = *dort* | *aime* | *a vu* | *a embauchée*

- Y a-t-il vraiment un besoin linguistique ?

  Le chat dort. Le chat que la cousine a vu dort

  ? Le chat que la voisine que le voisin aime a vu dort

  ** Le chat que la voisine que le voisin que la police recherche aime a vu dort*

- On ne peut pas supprimer les récursivités générales, mais ce n'est pas trop grave... Et on peut toujours fixer une limite aux imbrications de relatives, ex. 3 maximum.

Supprimer les récursivités

- Il vaut mieux développer des grammaires hors-contexte que des grammaires régulières pour nommer, gérer et combiner des grands nombres de règles
- Les algorithmes traitant des grammaires régulières sont bien plus rapides que les algorithmes traitant des grammaires hors-contexte
- Certains outils informatiques (ex. NooJ) peuvent supprimer les récursivités à gauche et à droite, et transformer les grammaires hors-contexte en grammaires régulières, et ainsi analyser les textes avec de bonnes performances.

Exercice

Construire une grammaire qui permet de reconnaître les noms de personne, comme par ex. :

M. Dupont, Mme Jeanne Dupont-Durand, Mlle de la Castellerie, Jeanne d'Artagnan

On pourra utiliser les symboles auxiliaires Prénom+m, Prénom+f et CAP

Personne = ...

Titre+m = ...

Titre+f = ...

Nomdefamille = ...

Exercice

Construire une grammaire qui permet de reconnaître les noms de personne, comme par ex. :

M. Dupont, Mme Jeanne Dupont-Durand, Mlle de la Castellerie, Jeanne d'Artagnan

On pourra utiliser les symboles auxiliaires Prénom+m, Prénom+f et CAP

Personne = (Titre+m|ε) Prénom+m Nomdefamille |

(Titre+f|ε) Prénom+f Nomdefamille

Titre+m = (M.|Monsieur)

Titre+f = (Mme | Madame | Mlle | Mademoiselle)

Nomdefamille = CAP (- CAP |ε) (de | d' | du | de la) CAP

Exercices à rendre

7. (1 pt) Construire la grammaire qui reconnaît toutes les phrases que l'on peut construire à partir du vocabulaire V = { Anne, voisine, donne, à, un, une, le, la, son, sa, cadeau, pomme}

8. (1 pt) Le mot anglais *that* a quatre fonctions potentielles : DETERMINANT, PRONOM, CONJONCTION ou ADVERBE. Sur le modèle de la grammaire p. 59, construire une grammaire pour le désambiguïser dans certains cas.

9. (2 pts) Construire la grammaire qui reconnait des dates en français :

Date = ...

Jour = *lundi* | *mardi* | *mercredi* | *jeudi* | *vendredi* | *samedi* | *dimanche*

Mois = (*janvier* | ...

... = ...

On pourra utiliser le symbole auxiliaire Nombre2-29.

La grammaire doit reconnaître : *lundi* ; *mardi 30 avril* ; *le 2 mai* ; le 31, mais ne doit pas reconnaître : ~~*vendredi 31 février*~~ ; ~~*samedi juin*~~ ; ~~27 mai~~ ; ~~*1 juillet*~~

4. Grammaires contextuelles

Grammaires génératives contextuelles (type 1)

- Les règles de réécriture peuvent contenir 2 symboles à gauche ; le premier doit être présent aussi dans le second membre, ex. :

SINGULIER PHRASE → SINGULIER GN *voit* GN

PLURIEL PHRASE → PLURIEL GN *voient* GN

GN → PLURIEL GN | SINGULIER GN

SINGULIER GN → SINGULIER le NOMSINGULIER

PLURIEL GN → PLURIEL les NOMPLURIEL

SINGULIER → ε

PLURIEL → ε

NOMSINGULIER → *chat*

NOMSINGULIER → *chien*

NOMPLURIEL → *chats*

NOMPLURIEL → *chiens*

Grammaires génératives contextuelles (Type 1)

- Les règles de réécriture peuvent contenir 2 symboles à gauche ; le premier doit être présent aussi dans le second membre, ex. :

SINGULIER PHRASE → SINGULIER GN *voit* GN

PLURIEL PHRASE → PLURIEL GN *voient* GN

GN → PLURIEL GN | SINGULIER GN

SINGULIER GN → SINGULIER le NOMSINGULIER

PLURIEL GN → PLURIEL les NOMPLURIEL

SINGULIER → ε

PLURIEL → ε

NOMSINGULIER → *chat*

NOMSINGULIER → *chien*

NOMPLURIEL → *chats*

NOMPLURIEL → *chiens*

Les symboles SINGULIER et PLURIEL sont utilisés pour marquer les contextes.

Langage décrit : { le chat voit le chat, le chat voit le chien, le chat voit les chats, le chat voit les chiens, le chien voit le chat, le chien voit le chien, le chien voit les chats, le chien voit les chiens, les chats voient le chat, les chats voient le chien, les chats voient les chats, les chats voient les chiens, les chiens voient le chat, les chiens voient le chien, les chiens voient les chats, les chiens voient les chiens }

Les séquences *les chats voit le chien* et *les chat voit les chien* ne sont pas reconnues

Grammaires contextuelles (Type 1), notation alternative

(ici : notation Lexical-Functional Grammar)

- Exemple de règle de grammaire LFG :

VP --> V: ↑=↓;
(NP: (↑OBJ)=↓
(↓CASE)=ACC)
PP*: ↓∈(↑ADJUNCT).

- Le constituant VP contient un verbe V, un objet optionnel (NP) et un nombre quelconque de compléments prépositionnels PP*
- « ↑=↓ » signifie que le constituant VP hérite des propriétés du verbe V
- « (↑OBJ)=↓ » signifie que le constituant VP hérite des propriétés de OBJ
- « (↓CASE)=ACC » signifie que le NP doit être à l'accusatif
- « ↓∈(↑ADJUNCT) » signifie que les compléments PP appartiennent à l'ensemble des modifieurs (« adjunct » en anglais) du VP

Grammaires contextuelles (Type 1), notation alternative

(ici : notation simplifiée)

- Exemple de grammaire hors contexte :

GNSINGULIER = DETSINGULIER (ε | ADJSINGULIER) NOMSINGULIER

GNPLURIEL = DETPLURIEL (ε | ADJPLURIEL) NOMPLURIEL

GN = GNSINGULIER | GNPLURIEL

- La grammaire contextuelle correspondante est beaucoup plus simple si elle ne contient pas les contraintes d'accord en nombre :

GN = DET (ε | ADJ) NOM

- On va donc ajouter à cette grammaire hors-contexte les contraintes d'accord

Grammaires contextuelles (Type 1), notation alternative

(ici : notation simplifiée)

- Grammaire contextuelle :

GN = **DET**

(ε | **ADJ**)

NOM

- Grammaire contextuelle = Grammaire hors contexte + on ajoute les contraintes d'accord

Grammaires contextuelles (Type 1), notation alternative

(ici : notation simplifiée)

- Grammaire contextuelle :

GN = **DET**/<**THIS**$Nombre=**N**$Nombre>

(ε | **ADJ**/<**THIS**$Nombre=**N**$Nombre>)

(**N** **NOM**)

- Grammaire contextuelle = Grammaire hors contexte + on ajoute les contraintes d'accord :

➢ des variables contiennent des ALU : **THIS** (ALU courante), **N**

➢ des propriétés morphologiques et lexicales : $Nombre, $Genre, $Cas, $Temps, $Classedistributionnelle, etc.

➢ des contraintes : <**THIS**$Nombre=**N**$Nombre>

Exemples de grammaires contextuelles morphologiques

(ici : notation simplifiée)

- Reconnaître des verbres dérivés en adjectifs en *–able* :

L = *a* | *b* | *c* | ... | *z*

ADJECTIF = **(Prefixe L L*** **)** able/**<Prefixe#er=:VERBE>**

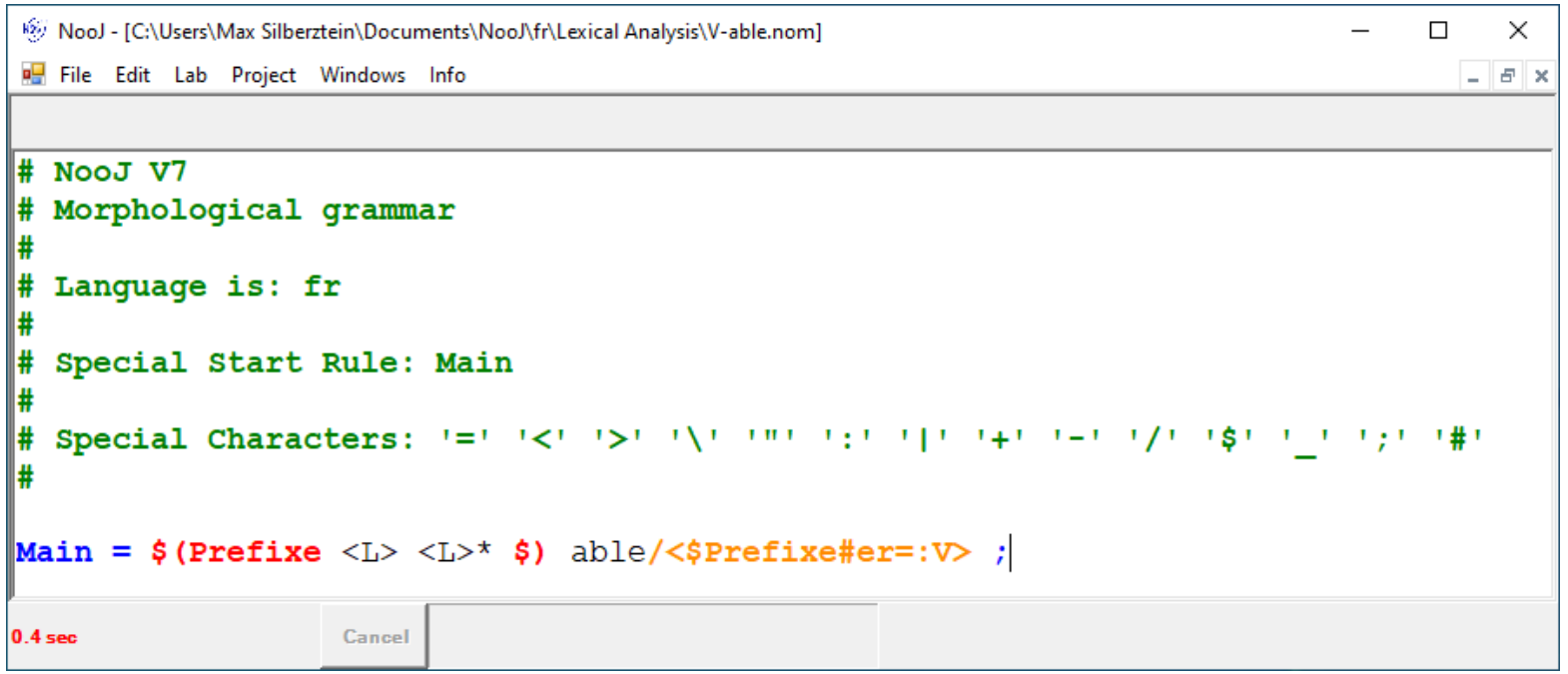
VERBE = *aimer* | *brancher* | *casser* | *donner* | *écouter* | *voler* | ...

Exemples de formes reconnues : *aimable, branchable, cassable*

La forme *table n'est pas reconnue car « ter » n'est pas un verbe*

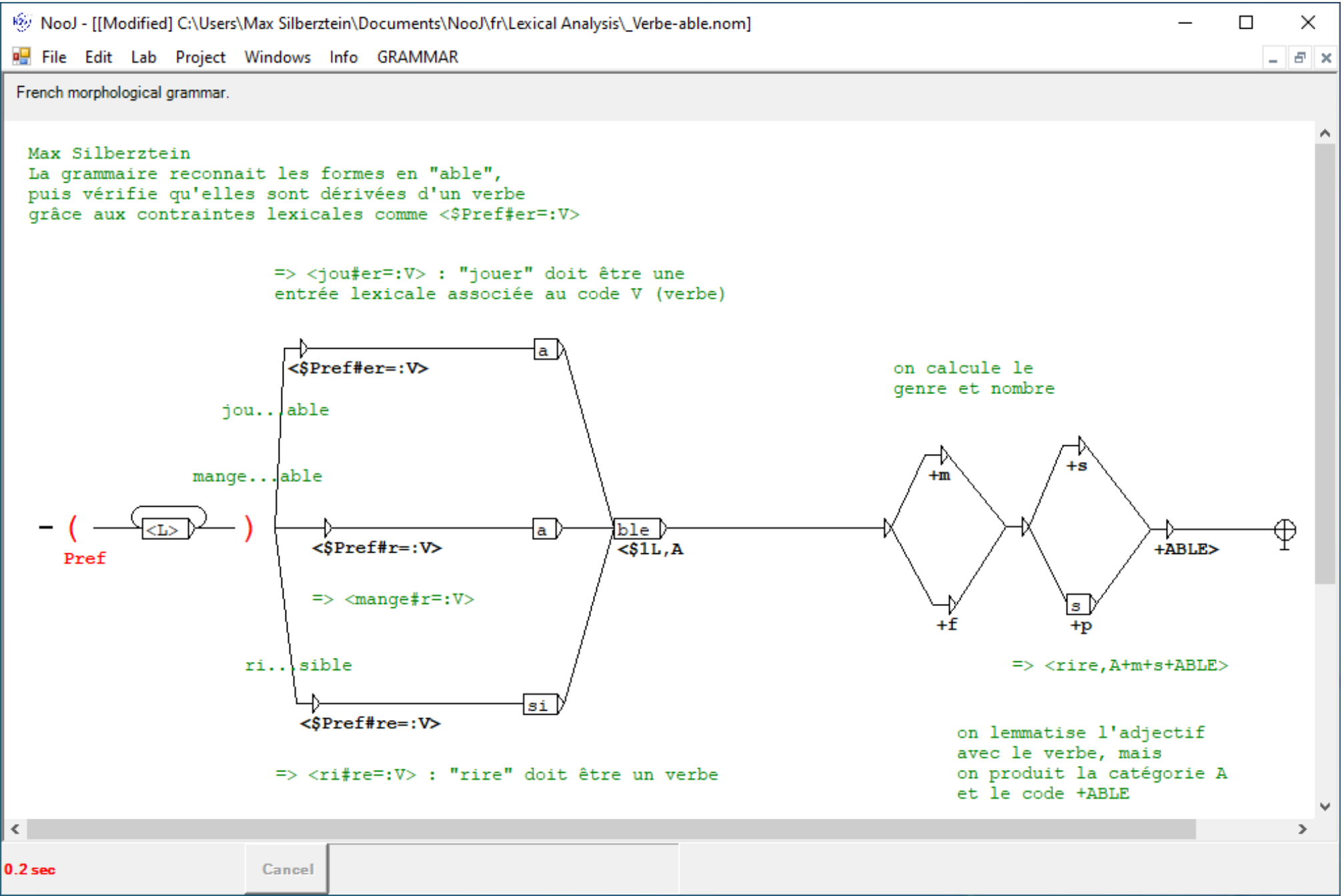
Exemples de grammaires contextuelles morphologiques

(ici : notation NooJ)



- Le symbole **Main** est le symbole auxiliaire initial
- Le symbole <L> représente n'importe quelle lettre ; <L><L>* représente toutes les séquences de lettres ; ces séquences sont rangées dans la variable $Prefixe.
- La contrainte <$Prefixe#er=:V> vérifie que si l'on concatène la suite de lettres rangée dans la variable $Prefixe avec le « er », on obtient un verbe.

Verbres dérivés en *–able* (notation graphique NooJ)



- On range toutes les lettres (<L>) du préfixe avant « able » dans la variable $Pref

- On vérifie les contraintes, ex. : <$Pref#er=:V> (on ajoute « er » après $Pref ; on vérifie que la forme obtenue est un verbe)

- On produit le résultat d'analyse, par exemple, les deux annotations pour *jouable* : <jouer,A+m+s+ABLE> <jouer,A+f+s+ABLE>

Exemples de grammaires contextuelles morphologiques

- Reconnaître des verbes préfixés en *re-* et *dé-* :

VERBEPREF = (r | re | ré | (ε|re) (dé|dés)) **(Suffixe L L*** **)/<Suffixe=:VERBE>**

Exemples de formes reconnues : *rappeler, remanger, réapprendre, redémonter*

Le verbe *rater n'est pas reconnu car « ater » n'est pas un verbe*

Exercice : grammaire contextuelle qui reconnait les phrases sémantiquement correctes :

- Propriétés des noms Nom :

$Nombre = singulier | pluriel

$Genre = masculin | féminin

$Classe = Hum | Conc | Abst

- Propriétés des verbes Verbe :

$Structure = Transitif| Intransitif | Pronominal

$Nombre = singulier| pluriel

$Classe0 = Hum | Conc | Abst

$Classe1 = Hum | Conc | Abst

Ajouter les contraintes d'accord, de temps et distributionnelle à la grammaire suivante :

Phrase = Sujet Verbe (ε|Objet)

GN = (le|la|les) Nom

Sujet = GN

Objet = GN

Verbe = ...

Grammaire contextuelle qui reconnait les phrases sémantiquement correctes :

On utilise trois variables :

- **S** et **V** pour l'accord en Nombre du sujet et de sa classe sémantique Classe avec les propriétés Nombre et Classe0 du verbe
- **N** pour l'accord en Genre et en Nombre du nom avec le déterminant
- Seuls les verbes transitifs peuvent avoir un complément d'objet direct ; contraintes sémantiques entre le verbe **V** et le sujet **S**, et entre le verbe **V** et l'objet **O** :

Phrase = Sujet/<S$Nombre=V$Nombre><S$Classe=V$Classe0>
Verbe
(ε | Objet/<V$Structure=Transitif><O$Classe=V$Classe1>)

GN = (le|la|les)/<THIS$Genre=N$Genre><THIS$Nombre=N$Nombre> ($_N$ Nom)

Sujet = ($_S$ GN)=N

Objet = ($_O$ GN)=N

Verbe = ($_V$ Verbe)

5. Grammaires non restreintes

Grammaires génératives non restreintes

- Aucune restriction : les membres gauches et les membres droits des règles de réécriture peuvent contenir n'importe quelle séquence d’ALUs, de symboles auxiliaires et de mots vides.

- Les grammaires non restreintes permettent de décrire n'importe quel langage récursivement énumérable (donc, n’importe quelle langue naturelle)

Grammaires génératives non restreintes

- Exemple de grammaire générative :

PHRASE → **GN0** mange **GN1**

GN0 mange **GN1** → **GN0** ne mange pas **GN1**

GN0 → Luc

GN1 → une pomme

GN0 mange **GN1** → C ’est **GN0** qui mange **GN1**

GN0 mange **GN1** → il mange **GN1**

GN0 mange **GN1** → **GN0** la mange

GN0 mange **GN1** → **GN1** est mangée par **GN0**

…

Grammaires génératives non restreintes alternatives
(notation HPSG)

- Exemple de grammaire HPSG qui représente la phrase « Kim relies on Sandy »

Représentation
de l'entité « Kim »

$$\left\langle \text{Kim} , \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{HEAD} & \textit{noun} \\ \text{VAL} & \begin{bmatrix} \text{SPR} & \langle\,\rangle \\ \text{COMPS} & \langle\,\rangle \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{INDEX} & \text{x} \\ \text{RELS} & \left\langle \begin{bmatrix} \text{PRED} & \text{named} \\ \text{ARG0} & \text{x} \\ \text{NAME} & \text{“Kim”} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix} \right\rangle$$

Représentation
de l'entité « Sandy »

$$\left\langle \text{Sandy} , \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{HEAD} & \textit{noun} \\ \text{VAL} & \begin{bmatrix} \text{SPR} & \langle\,\rangle \\ \text{COMPS} & \langle\,\rangle \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{INDEX} & \text{x} \\ \text{RELS} & \left\langle \begin{bmatrix} \text{PRED} & \text{named} \\ \text{ARG0} & \text{x} \\ \text{NAME} & \text{“Sandy”} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix} \right\rangle$$

Représentation
du prédicat « relies on »

$$\left\langle \text{rely} , \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{HEAD} & \textit{verb} \\ \text{VAL} & \begin{bmatrix} \text{SPR} & \langle\ \text{NP[INDEX x]}\ \rangle \\ \text{COMPS} & \langle\ \text{PP[INDEX y]}\ \rangle \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{INDEX} & \text{e} \\ \text{RELS} & \left\langle \begin{bmatrix} \text{PRED} & \text{rely\_on} \\ \text{ARG0} & \text{e} \\ \text{ARG1} & \text{x} \\ \text{ARG2} & \text{y} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix} \right\rangle$$

Représentation
du mot « on »

$$\left\langle \text{on} , \begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{HEAD} & \textit{prep} \\ \text{VAL} & \begin{bmatrix} \text{SPR} & \langle\,\rangle \\ \text{COMPS} & \langle\ \text{NP[INDEX x]}\ \rangle \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{INDEX} & \text{x} \\ \text{RELS} & \langle\,\rangle \end{bmatrix} \end{bmatrix} \right\rangle$$

Représentation
de la structure de phrase

Head-Complement Rule

$$\begin{bmatrix} \textit{phrase} \\ \text{COMPS} & \langle\,\rangle \end{bmatrix} \rightarrow \mathbf{H} \begin{bmatrix} \textit{word} \\ \text{COMPS} & \langle\ \boxed{1}\ , \ldots, \boxed{n}\ \rangle \end{bmatrix} \quad \boxed{1} \ldots \boxed{n}$$

Head-Specifier Rule

$$\begin{bmatrix} \textit{phrase} \\ \text{SPR} & \langle\,\rangle \end{bmatrix} \rightarrow \boxed{1} \quad \mathbf{H} \begin{bmatrix} \text{SPR} & \langle\ \boxed{1}\ \rangle \\ \text{COMPS} & \langle\,\rangle \end{bmatrix}$$

Règles de combination des matrices

Semantic Compositionality Principle

$$\begin{bmatrix} \text{RELS} & \boxed{A_1} \oplus \ldots \oplus \boxed{A_n} \end{bmatrix} \rightarrow \begin{bmatrix} \text{RELS} & \boxed{A_1} \end{bmatrix} \ldots \begin{bmatrix} \text{RELS} & \boxed{A_n} \end{bmatrix}$$

Semantic Inheritance Principle

$$\begin{bmatrix} \text{INDEX} & \boxed{1} \end{bmatrix} \rightarrow \ldots \mathbf{H} \begin{bmatrix} \text{INDEX} & \boxed{1} \end{bmatrix} \ldots$$

Grammaires génératives non restreintes alternatives
(notation simplifiée)

- Exemple (sans les contraintes d'accord) :

NEGATION :

GN0 mange **GN1** = **GN0**/**THIS** <E>/ne mange/**THIS** <E>/pas **GN1**/**THIS**

Le voisin mange la pomme → Le voisin ne mange pas la pomme

EXTRACTION_0 :

GN0 mange **GN1** = <E>/C'est **GN0**/**THIS** <E>/qui mange/**THIS GN1**/**THIS**

Le voisin mange la pomme → C'est le voisin qui mange la pomme

PASSIF_FS :

GN0 mange **GN1** = **GN0**/**GN1** <E>/est <V>/**THIS**$Participepassé+f+s **GN1FS**/par **GN0**

Le voisin mange la pomme ⟶ la pomme est mangée par le voisin

…

Analyses sémantiques

- Notation prédicative utilisée en traitement des connaissances :

PRED :

(Acteur GN0) (Prédicat mange**) (Objet GN1) /Prédicat(Acteur, Agent)**

Le voisin mange la pomme → mange(Le voisin, la pomme)

- Notation RDF/Turtle (utilisée dans les applications de Web sémantique) :

TURTLE :

(Acteur GN0) (Prédicat mange**) (Agent GN1)**

<E>/<http://www.w3.org/TR/rdf-syntax-grammar>

<http://www.w3.org/**Acteur**>

<http://www.w3.org/**Prédicat**>

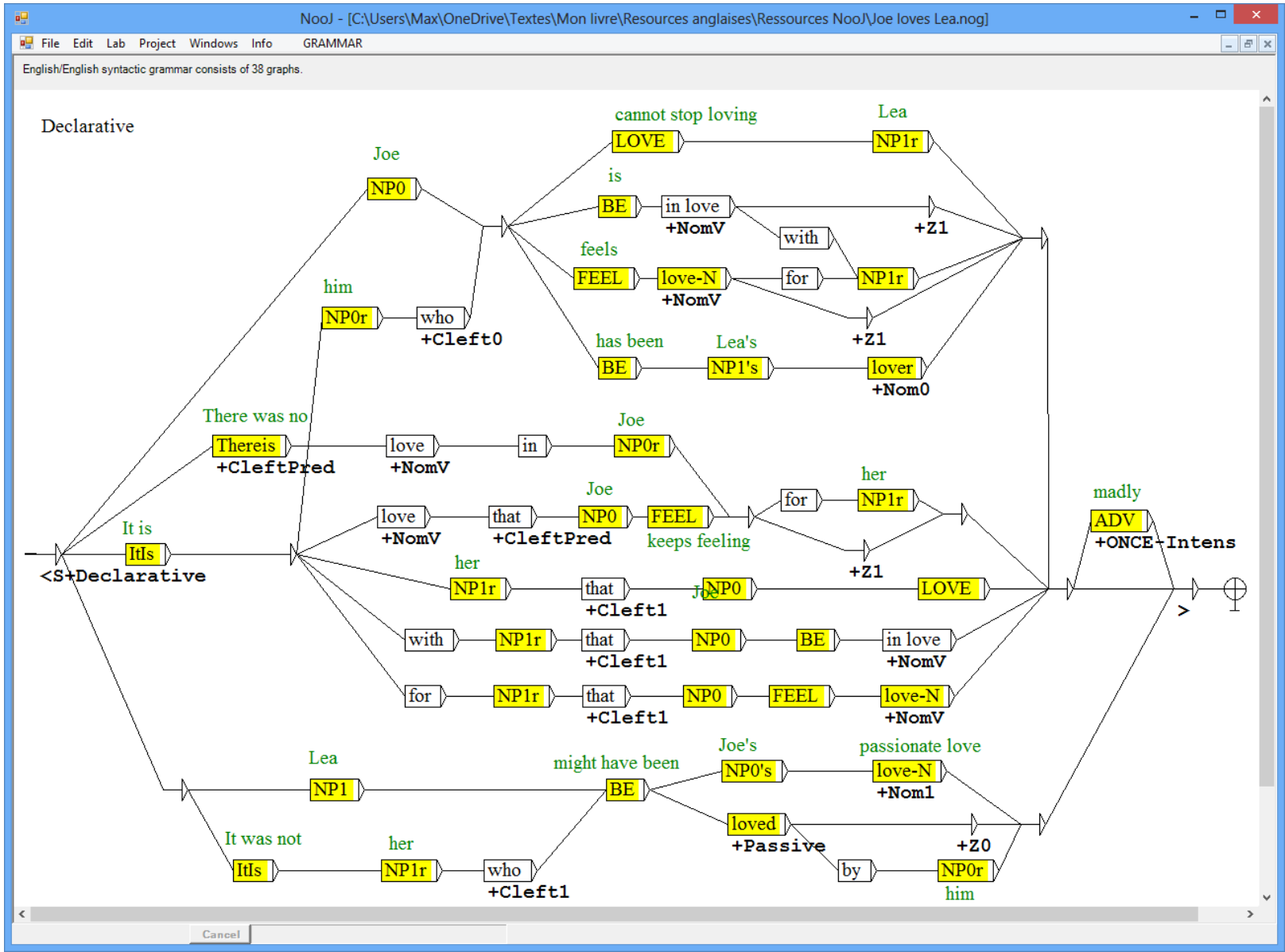
<http://www.w3.org/**Agent**>

Exemple de grammaire non restreinte
(notation graphique NooJ)

Les noeuds en jaune sont des références à des graphes imbriqués ; ils correspondent aux symboles auxiliaires des grammaires génératives.

Cette grammaire contient 38 graphes

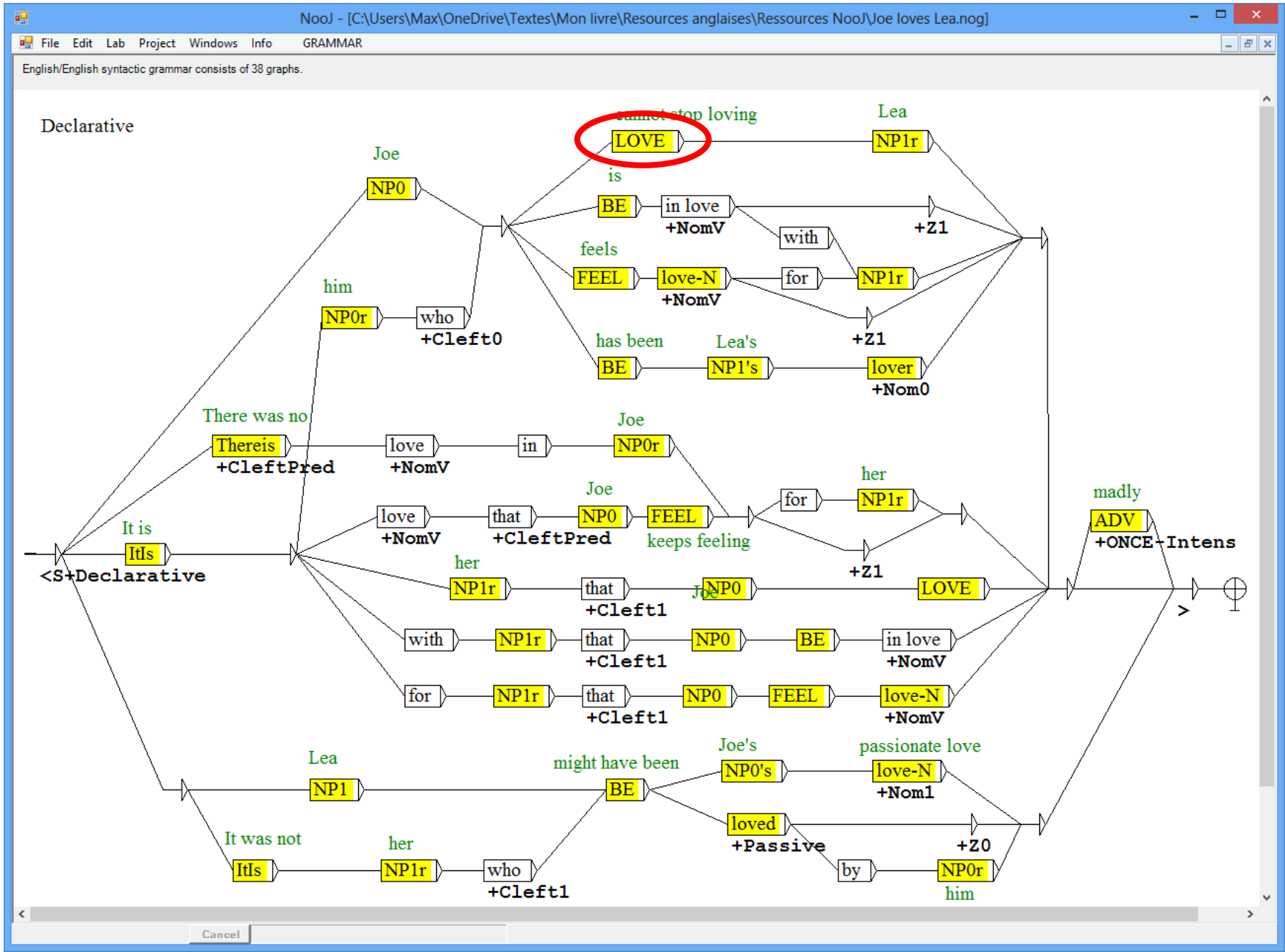
En général, les linguistes préfèrent utiliser la notation graphique lorsque les grammaires deviennent complexes.



Exemple de grammaire transformationnelle sous NooJ (notation graphique)

Les noeuds en jaune sont des références à des graphes imbriqués ; ils correspondent aux symboles auxiliaires des grammaires génératives

Le graphe LOVE est présenté ci-après

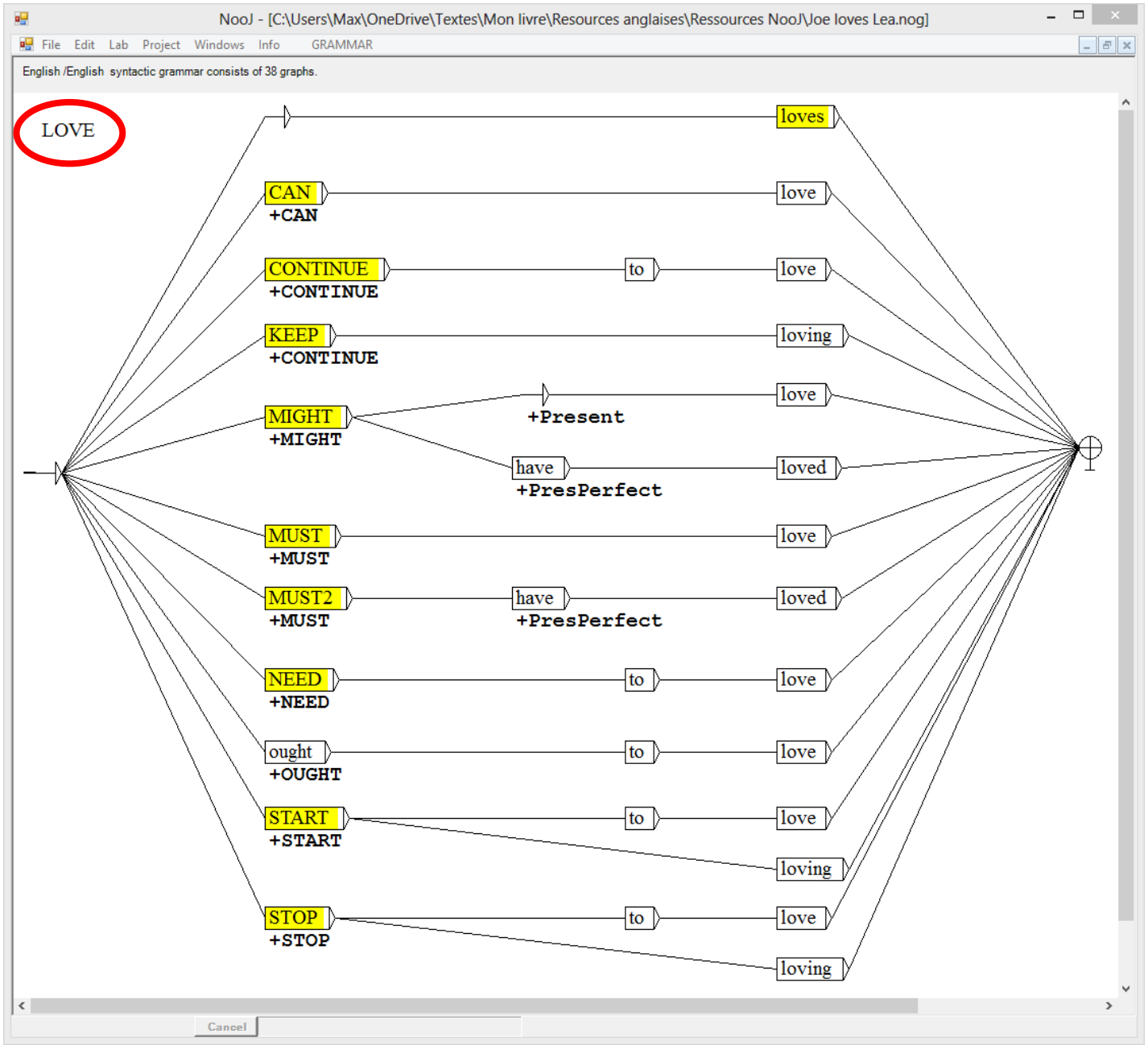


Exemple de grammaire transformationnelle sous NooJ (notation graphique)

Le graphe présenté ici est le graphe LOVE

Cette grammaire décrit les séquences constituées d'une forme conjuguée du verbe *to love* et des formes avec verbes auxiliaires, aspectuels et modaux, ainsi que les négations éventuelles, ex. :

(The butcher) must not have loved (his wife)



Exemple de grammaire transformationnelle sous NooJ (notation graphique)

On peut appliquer cette grammaire pour produire automatiquement toutes les phrases transformées à partir d'une phrase donnée, ici :

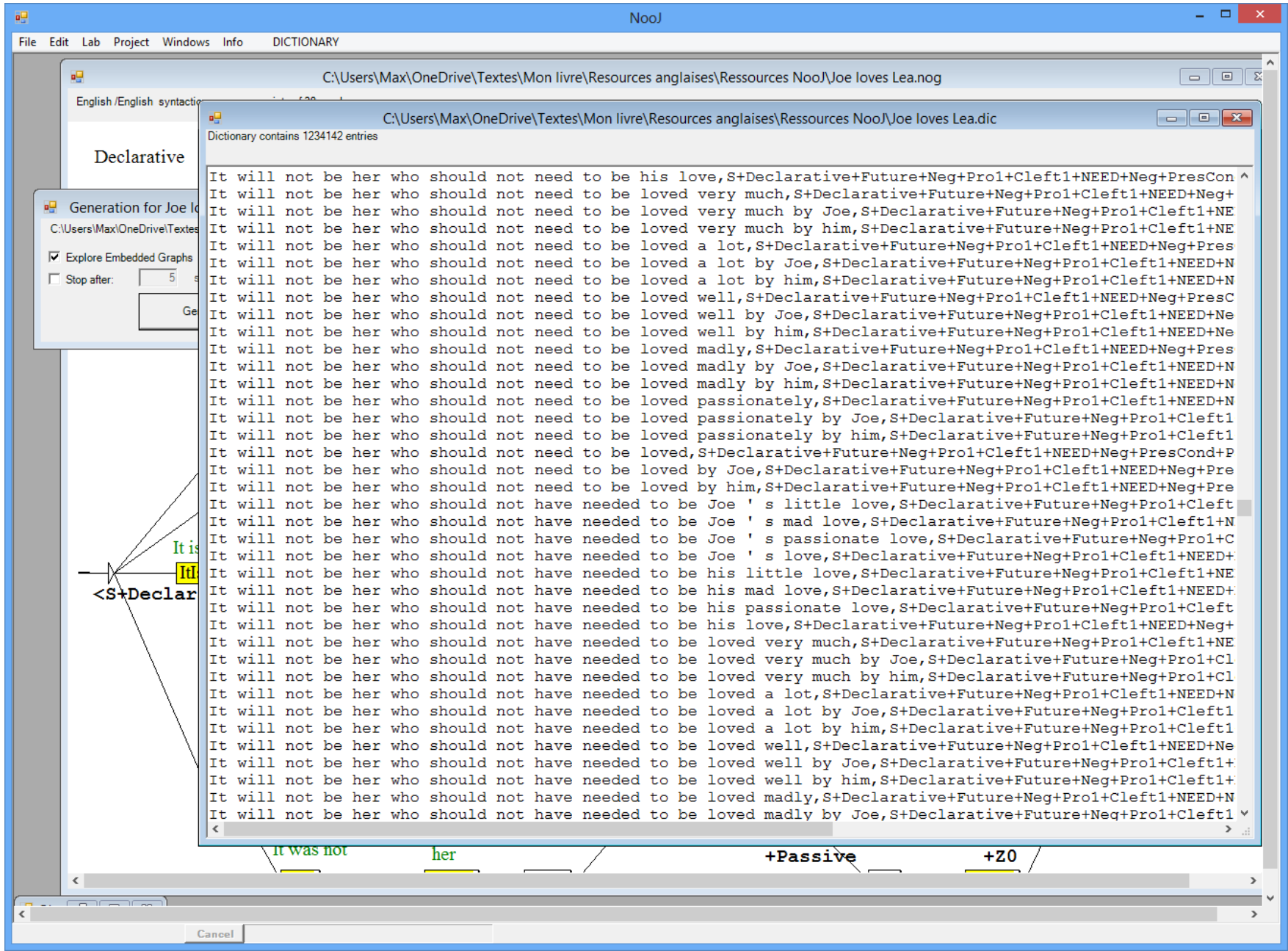
Joe loves Lea

Chaque phrase transformée est associées à une série de codes sémantiques, ex. : *It is Joe who did not love her* est associée à : +Declarative+Past+Neg+Pron1+Focus0

Certaines phrases sont stylistiquement discutables, ex. :

It will not be her who should not need to be loved by Joe.

On pourra ajouter des contraintes stylistiques sur la taille des phrases, le nombre de négation, limiter le nombre de verbes auxiliaires dans les phrases passives, etc.



Exercices à rendre

10. (2 pts) Construire une grammaire qui reconnaît des phrases qui contiennent une négation, une opinion ou un sentiment négatif, ex. :

Stéphane est un looser ; Marie n'est pas une gagnante

Stupidement, Jeanne est repartie ; Eric a peur de son ombre

On pourra utiliser les symboles auxiliaires NomPositif et NomNégatif. On donnera quelques exemples de verbes et adverbes positifs ou négatifs, ex. *stupidement*, *intelligemment*, *gagner*, *perdre*

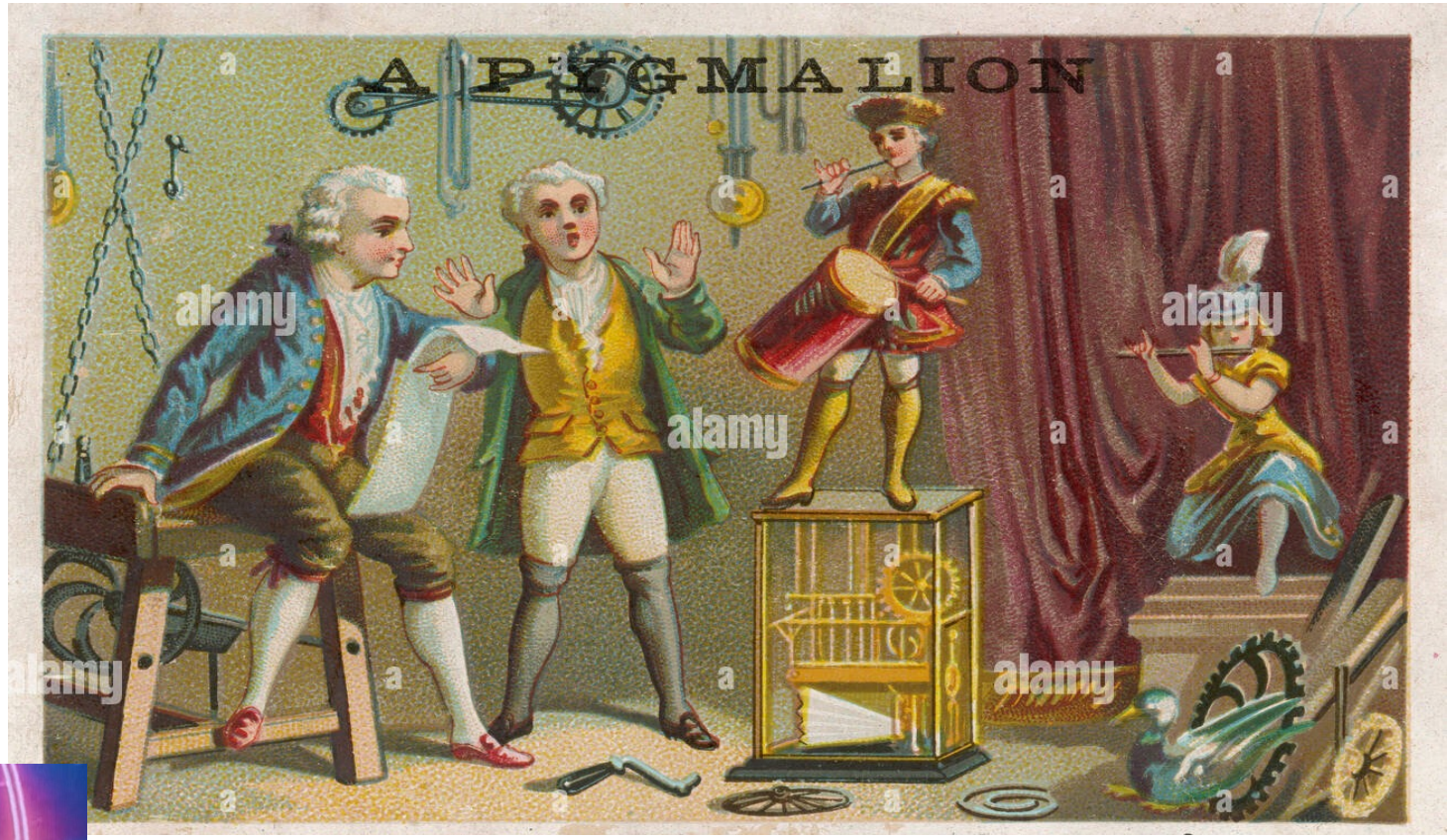
Automates

Τάλως

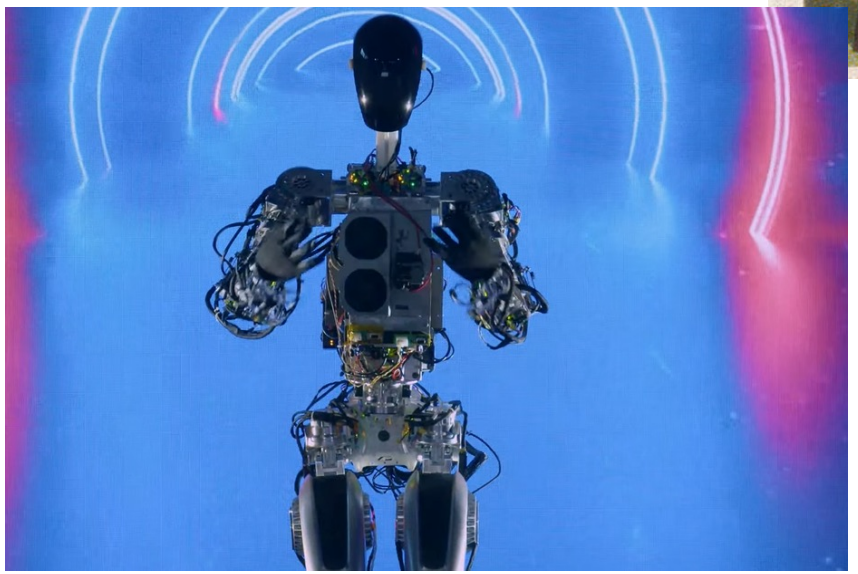
גּוֹלֶם

Le Turc

Horloge astronomique de Strasbourg, XIVe



Automates de Vaucanson, 1738-1782



Robot Tesla, 2022

Automates

Formalisation du comportement des automates

- Un automate est un mécanisme qui
  - est dans un certain état
  - accepte des signaux entrants
  - change d'état en fonction des signaux.

- Exemple d'automate :
  - une lampe est éteinte ; on presse un bouton => elle s'allume ; on represse le bouton => elle s'éteint

  - Poupée interactive : on appuie sur sa main gauche => elle gazouille ; on appuie sur son pied gauche => elle rit ; on effleure son pied droit => elle éternue ; ou elle fait des vocalises ; on lui donne son biberon => elle tète (bruit de succion) ; etc.

Dès 2 ans

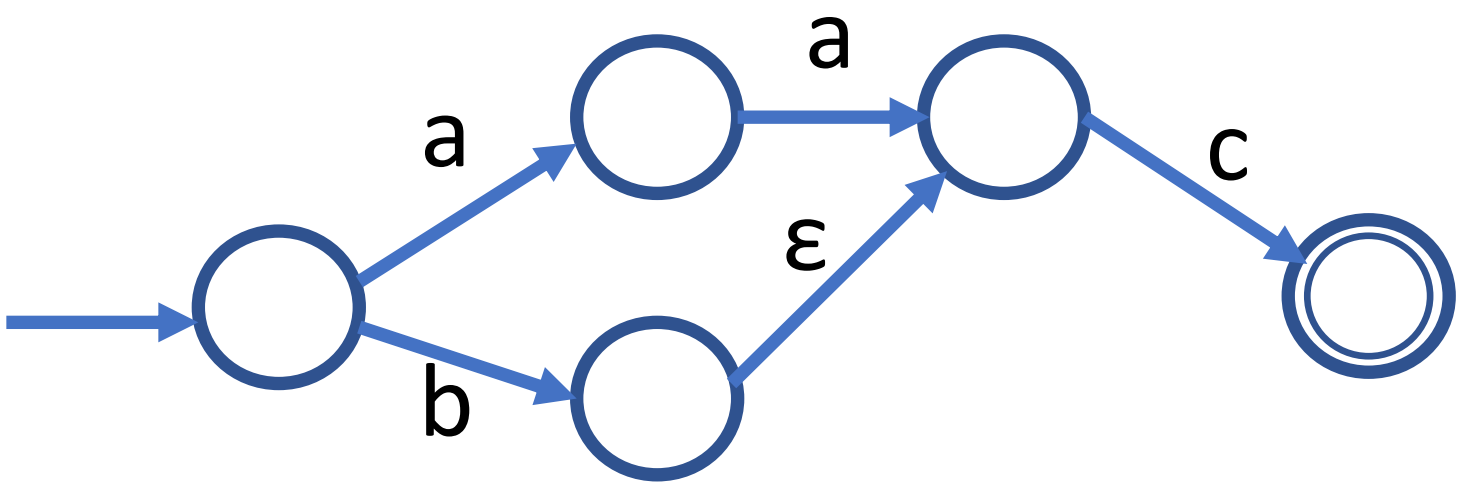
Poupon Lucille Interactive

(76) *En stock*

Formalisation du comportement des automates

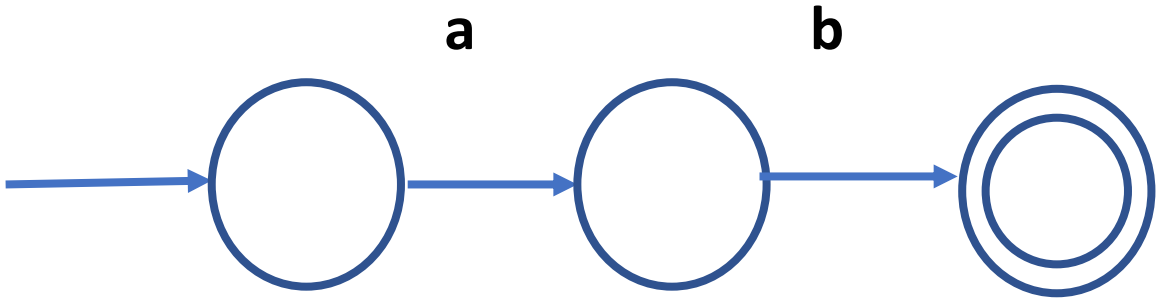
- un alphabet A
- un ensemble fini d'états Q
- un ensemble non vide d'états initiaux I
- un ensemble non vide d'états terminaux T
- un ensemble de transitions, c'est-à-dire de triplets (q1, l, q2) tels que q1 et q2 sont des états de Q, et l est une lettre de A

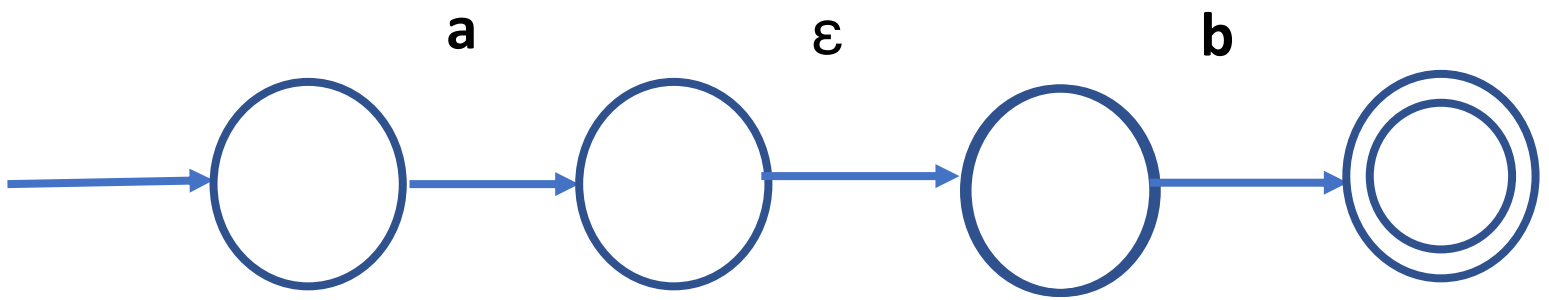
Automate fini : 5 états, 5 transitions



Un automate reconnaît tous les mots épelés par un chemin qui part d'un état initial et qui rejoint un état terminal.

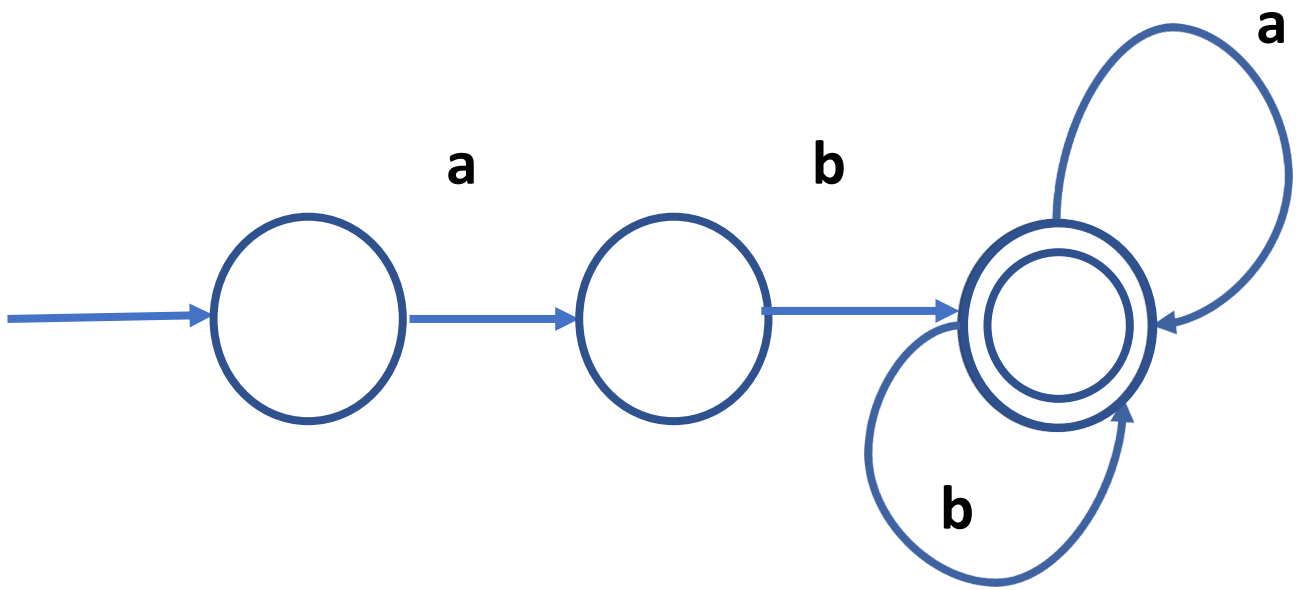
Sur l’alphabet {a, b}, construire l’automate fini qui reconnaît le mot « ab »





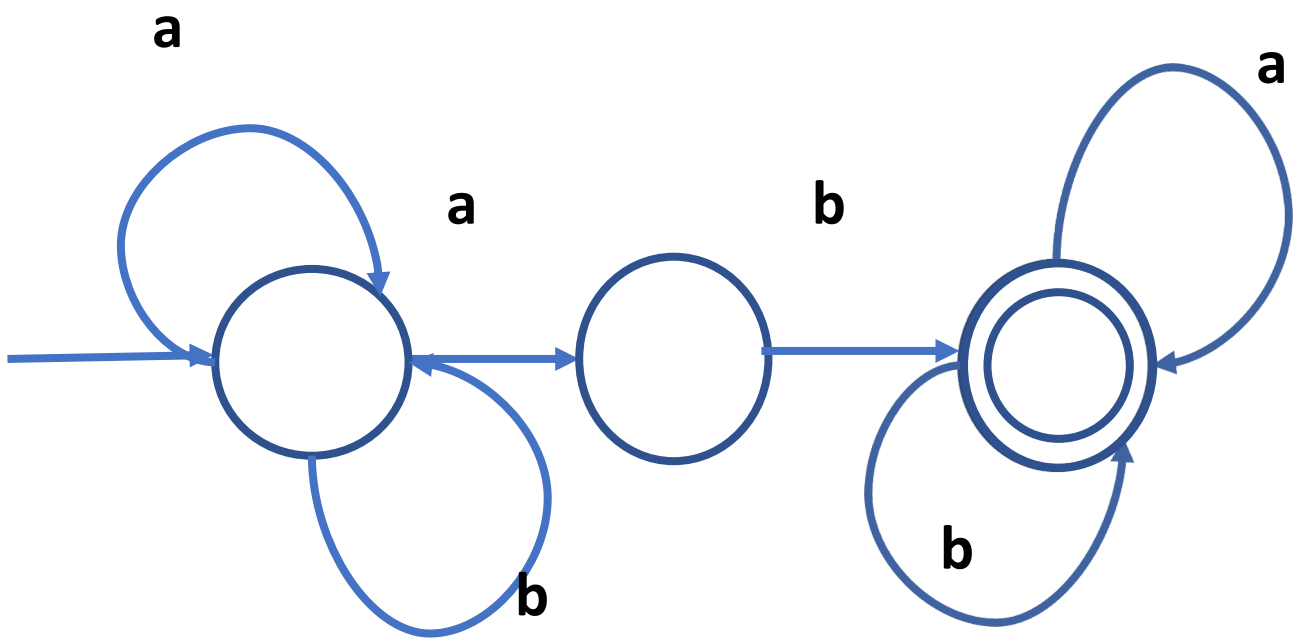
Sur l’alphabet {a, b}, construire l’automate fini qui reconnaît tous les mots qui commencent par « ab »

Sur l’alphabet {a, b}, construire l’automate fini qui reconnaît tous les mots qui commencent par « ab » (mais qui peuvent être suivis par n’importe quelle séquence de lettres)



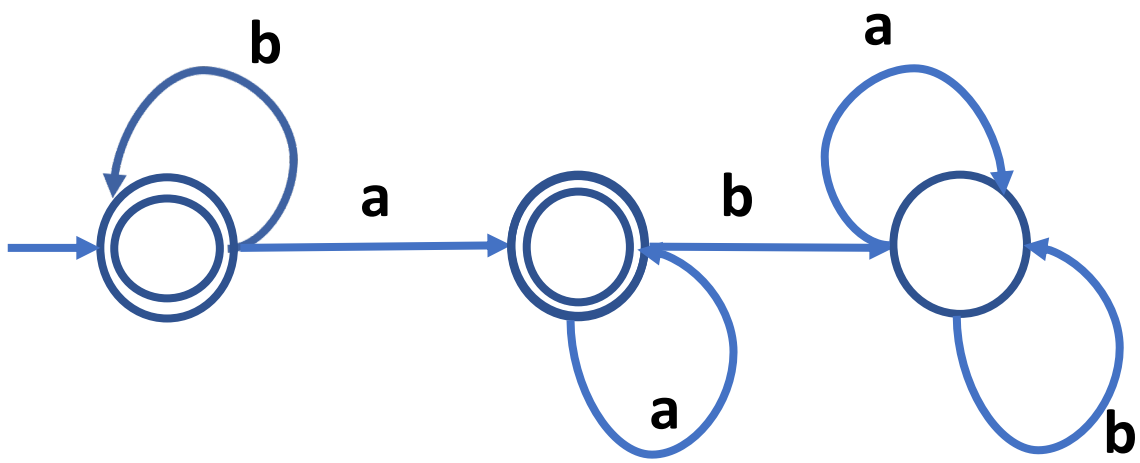
Sur l’alphabet {a, b}, construire l’automate fini qui reconnaît tous les mots qui contiennent l’affixe « ab »

Sur l’alphabet {a, b}, construire l’automate fini qui reconnaît tous les mots qui contiennent l’affixe « ab »



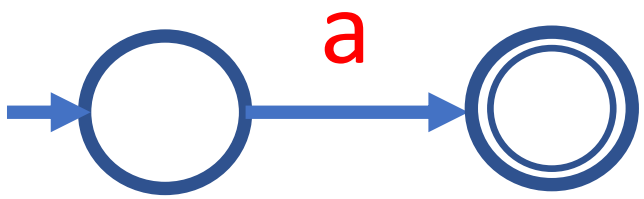
Sur l’alphabet {a, b}, construire l’automate fini qui reconnaît tous les mots qui ne contiennent pas l’affixe « ab »

Sur l’alphabet {a, b}, construire l’automate fini qui reconnaît tous les mots qui ne contiennent pas l’affixe « ab »

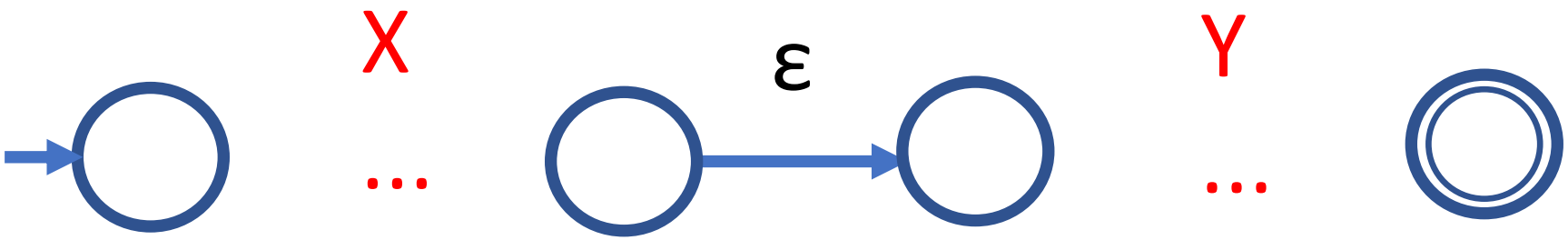


Construction de Thompson (1955)

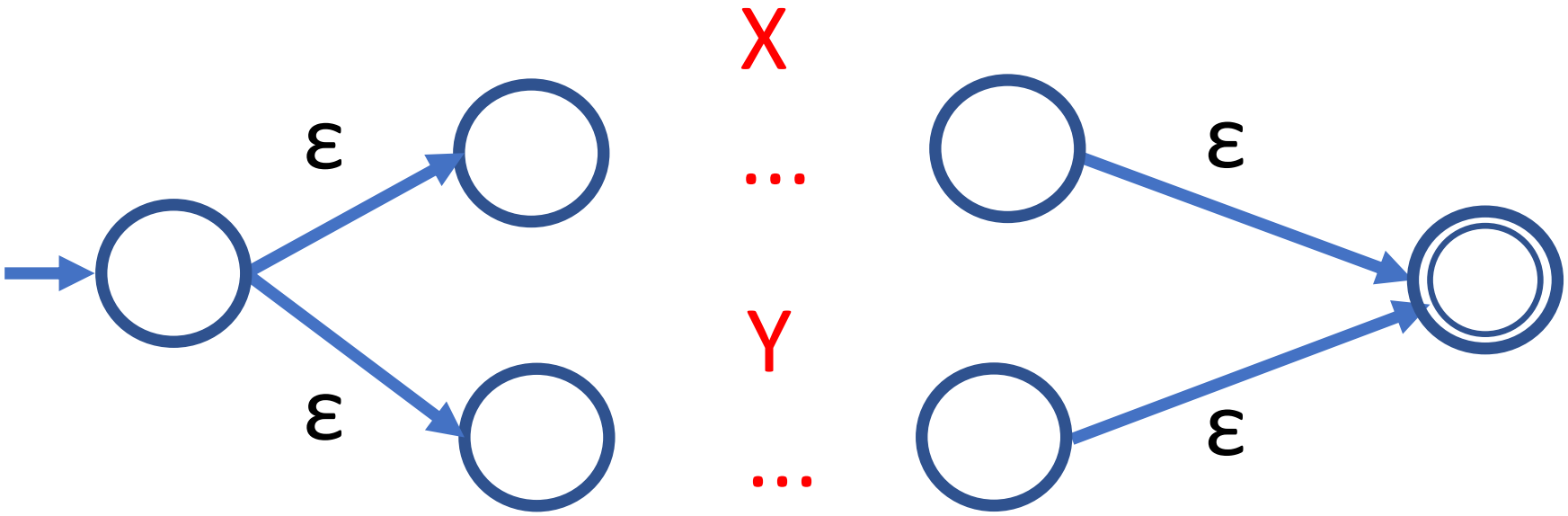
- Pour chaque lettre a, on construit l'automate :



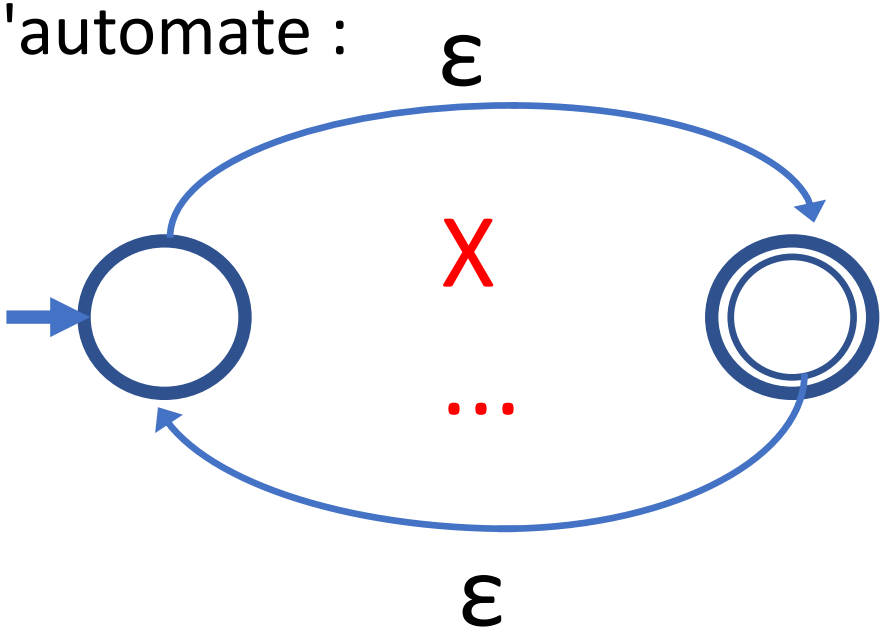
- Pour chaque concaténation X Y, on construit l'automate :



- Pour chaque disjonction X | Y, on construit l'automate :



- Pour chaque opération de Kleene X*, on construit l'automate :



Construction de Thompson

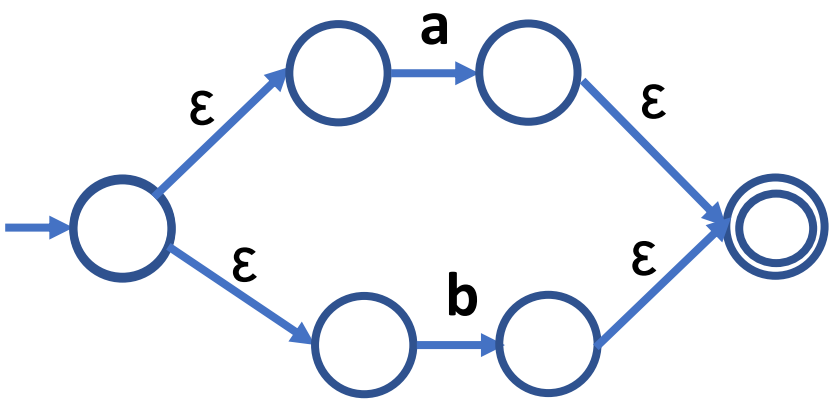
- Sur l’alphabet {a, b, c}, construire en suivant la méthode de Thompson l’automate fini pour l’expression régulière :

a | b

Construction de Thompson

- Sur l'alphabet {a, b, c}, construire en suivant la méthode de Thompson l'automate fini pour l'expression régulière :

a | b



Construction de Thompson

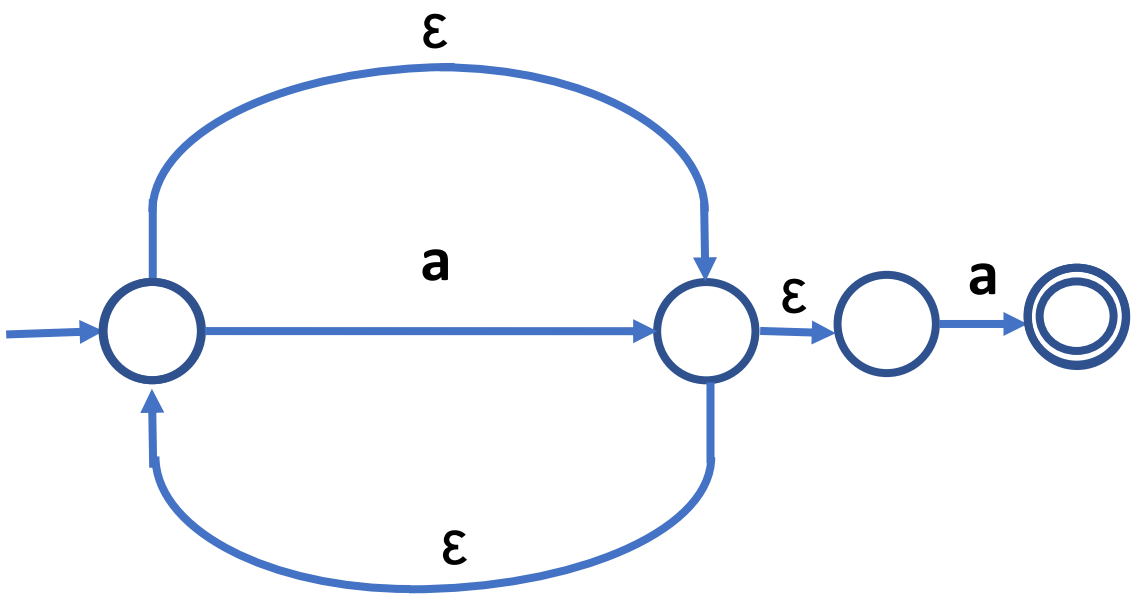
- Sur l’alphabet {a, b, c}, construire en suivant la méthode de Thompson l’automate fini pour l’expression régulière :

a* a

Construction de Thompson

- Sur l'alphabet {a, b, c}, construire en suivant la méthode de Thompson l'automate fini pour l'expression régulière :

a* a



Construction de Thompson

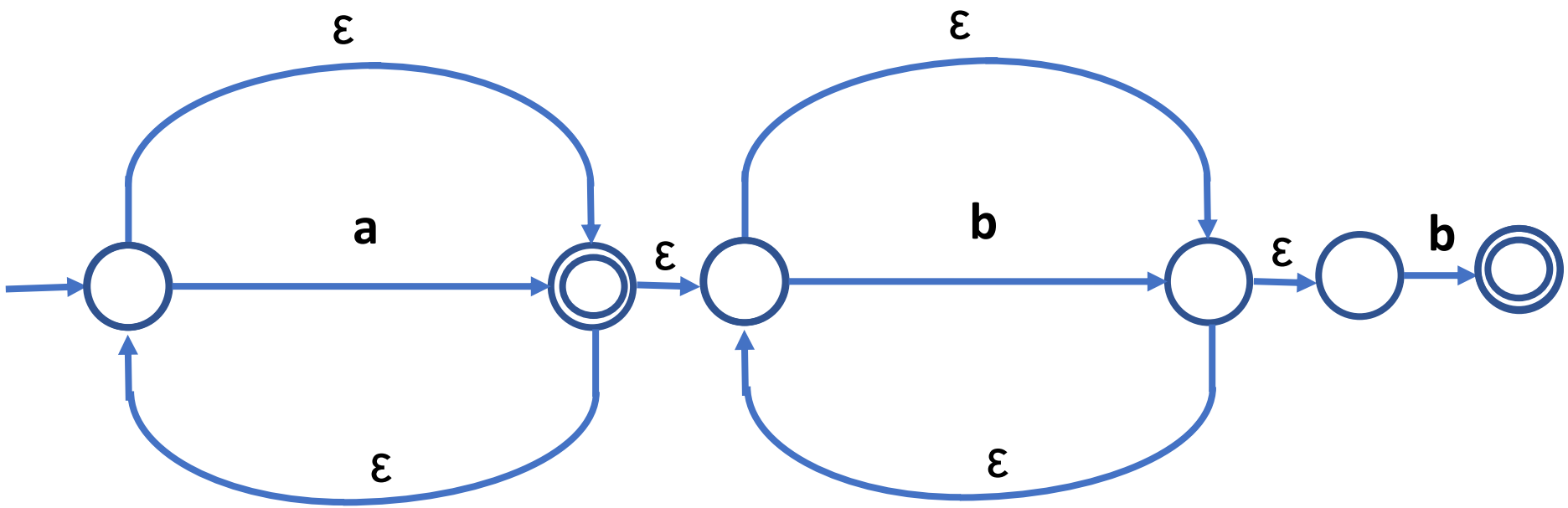
- Sur l'alphabet {a, b, c}, construire en suivant la méthode de Thompson l'automate fini pour l'expression régulière :

a* b* b

Construction de Thompson

- Sur l'alphabet {a, b, c}, construire en suivant la méthode de Thompson l'automate fini pour l'expression régulière :

a* b* b



ATTENTION : on ne peut pas unifier l'état terminal de l'automate gauche avec l'état initial de l'automate droit car dans ce cas, l'automate reconnaîtrait « abab », ce qui n'est pas décrit par la grammaire

Construction de Thompson

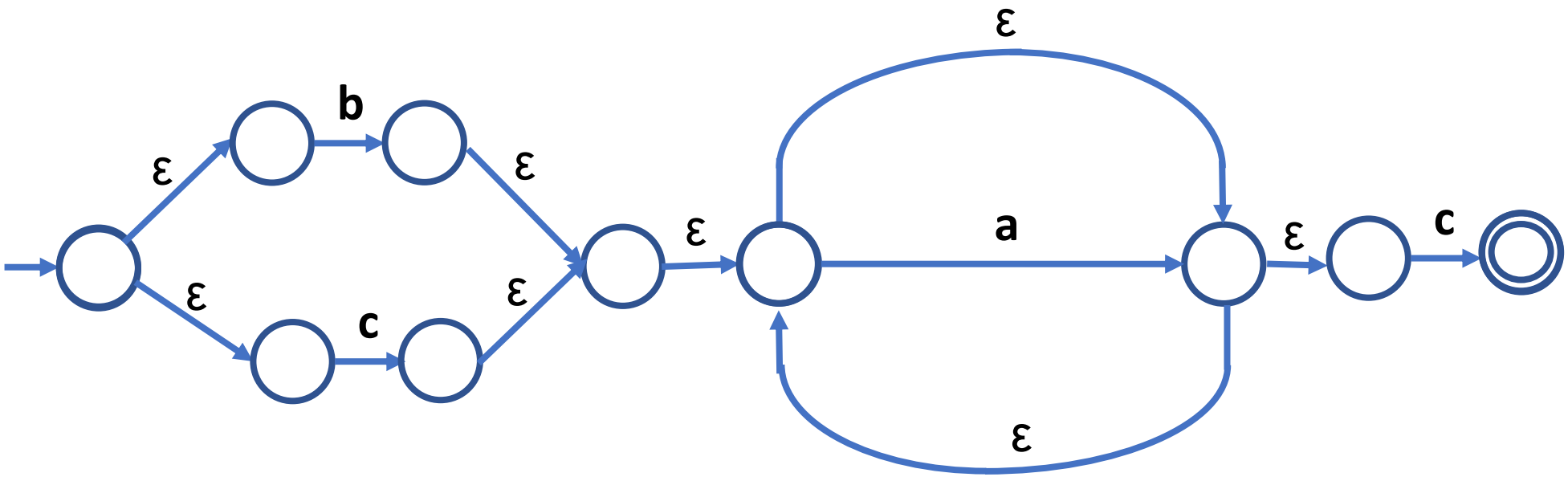
- Sur l’alphabet {a, b, c}, construire en suivant la méthode de Thompson l’automate fini pour l’expression régulière :

(b | c) a* c

Construction de Thompson

- Sur l'alphabet {a, b, c}, construire en suivant la méthode de Thompson l'automate fini pour l'expression régulière :

(b | c) a* c

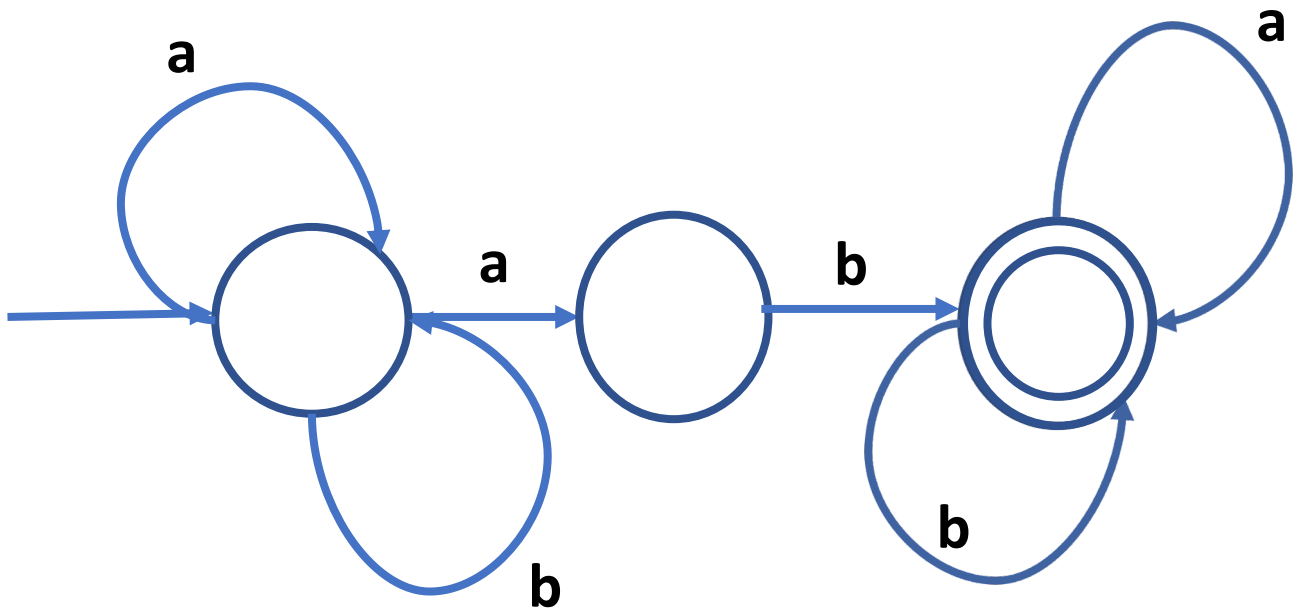


Automates déterministes

- Un automate est déterministe si :
  - il ne contient aucune transition étiquetée par ε
  - pour chaque état de l'automate, il n'existe au plus qu'une seule transition sortante étiquetée par une lettre donnée

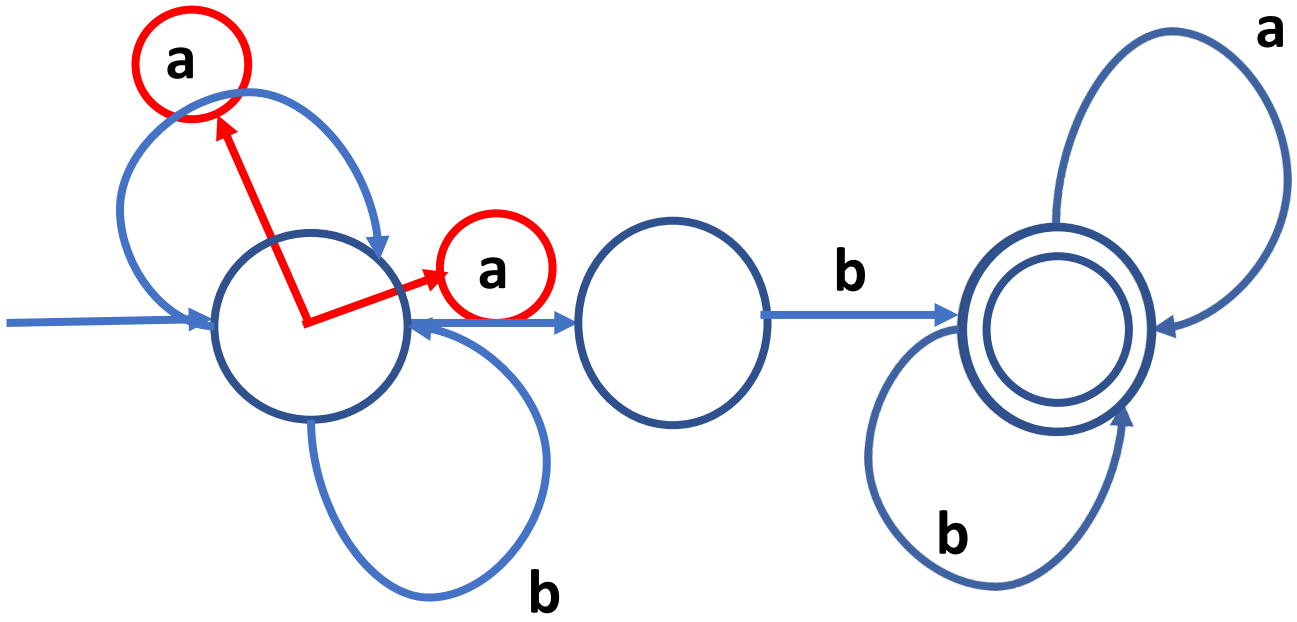
- On peut déterminiser tout automate

Automate fini déterministe ou non-déterministe ?

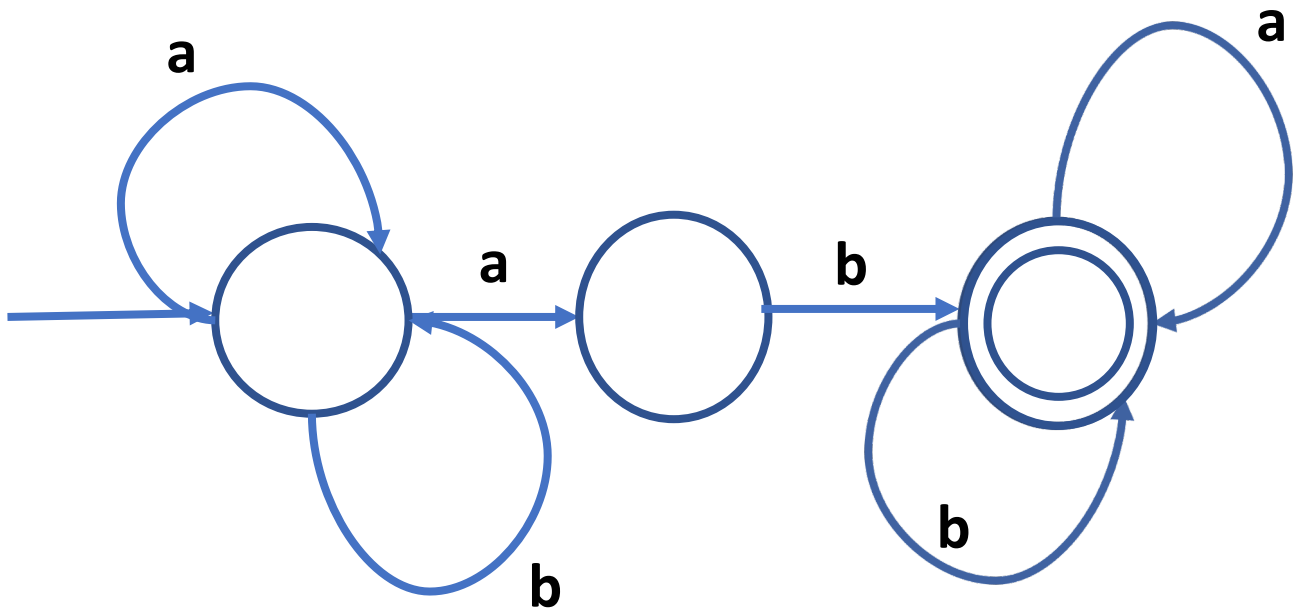


Automate fini non-déterministe :

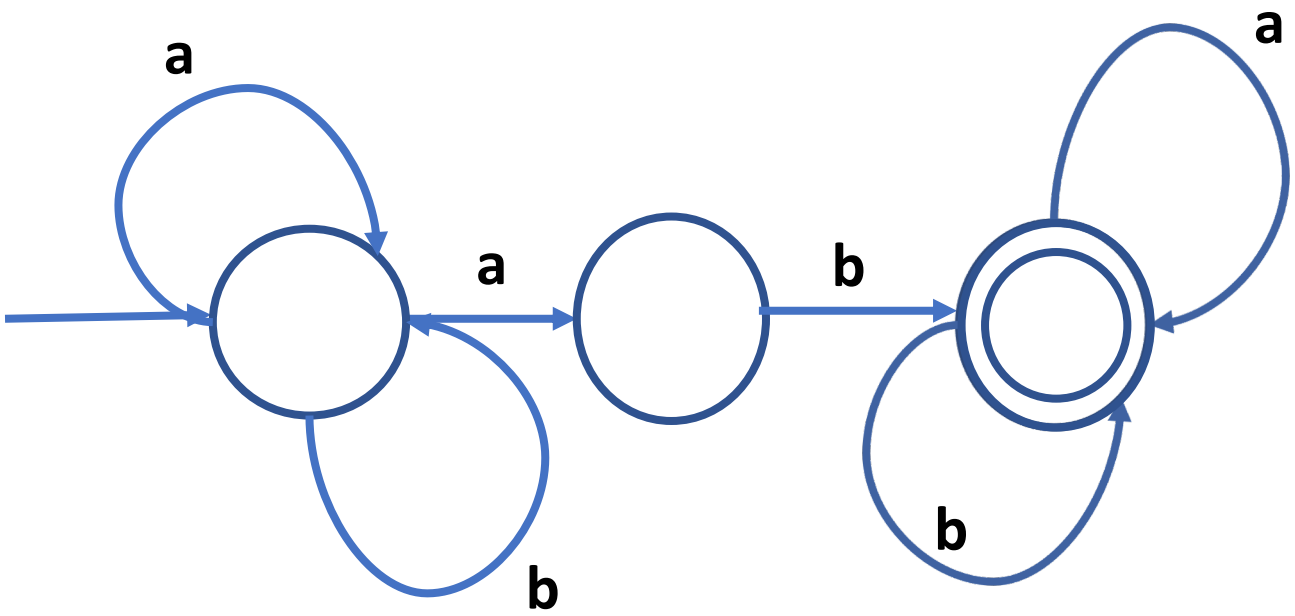
l'état initial contient 2 transitions sortante étiquetée par la même lettre a

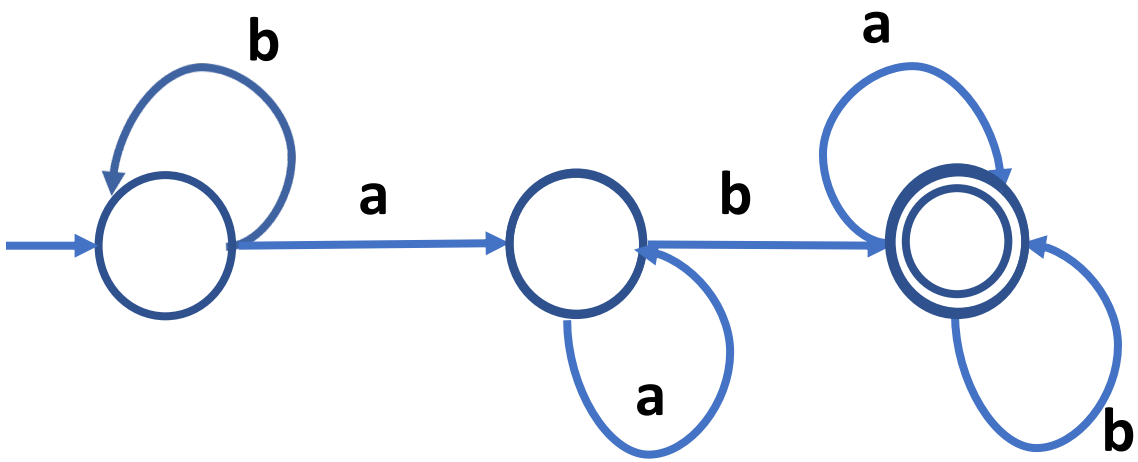


Exercice : construire l'automate fini déterministe équivalent, *i.e.*, qui reconnaît tous les mots qui contiennent l'affixe « ab »



Automate déterministe équivalent





Exercices à rendre

11. (1 pt) Sur l'alphabet {a, b}, construire l'automate déterministe qui reconnaît tous les mots qui se terminent par « ab »
12. (1 pt) Sur l'alphabet {a, b, c}, construire l'automate déterministe pour l'expression :

((a | b)* c)*

Sur l'alphabet français :

13. (1 pt) Construire l'automate déterministe qui reconnaît toutes les variantes orthographiques et morphologiques du nom *tsar* (cf. exercice corrigé en cours)

14. (2 pts) Compléter la grammaire des noms de personne (cf. exercice corrigé en cours) pour reconnaître des séquences qui contiennent des titres et leur abréviation (ex. *Général*, *Monseigneur*, *Docteur*, etc.)

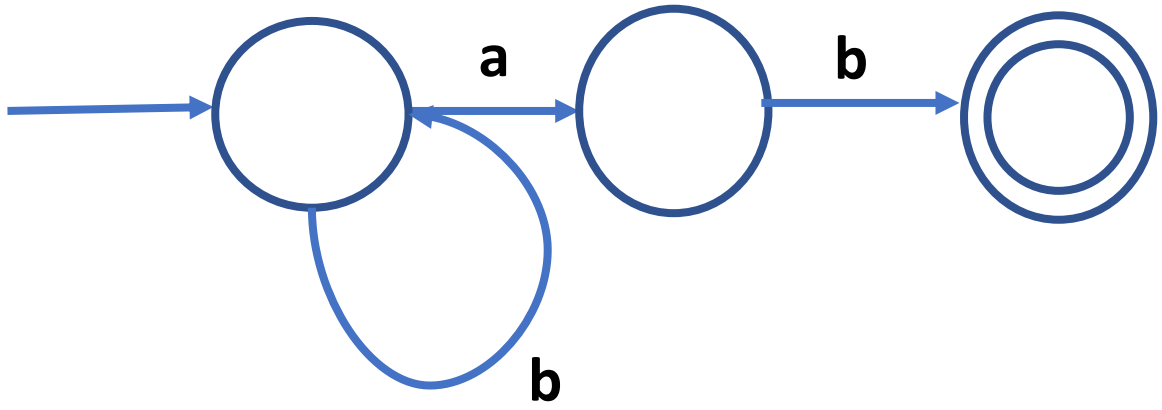
15. (1 pt) Construire l'automate qui reconnaît toutes les formes dérivées du verbe *monter* qui sont des lemmes (non fléchis)

Automates complets

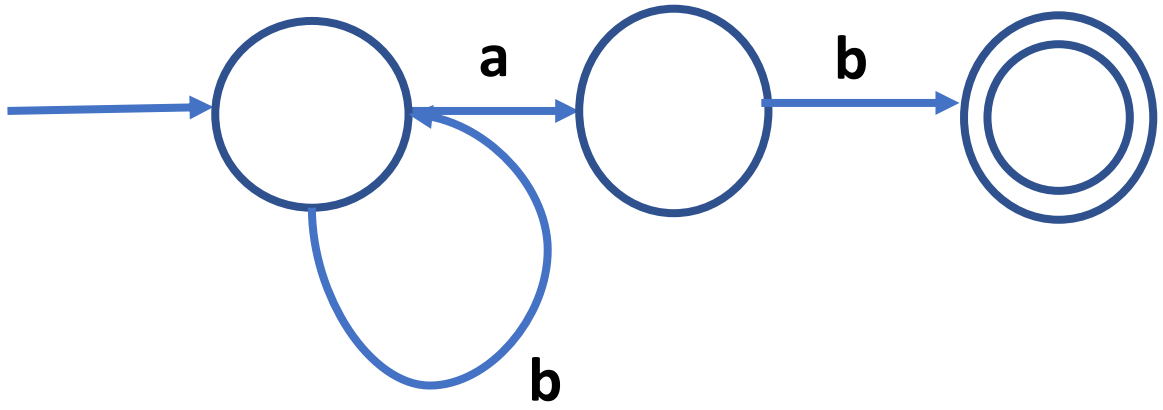
- Un automate est complet si :
  - il est déterministe
  - pour chaque état de l'automate, il existe une transition sortante étiquetée par chaque lettre de l'alphabet

- On peut compléter tout automate

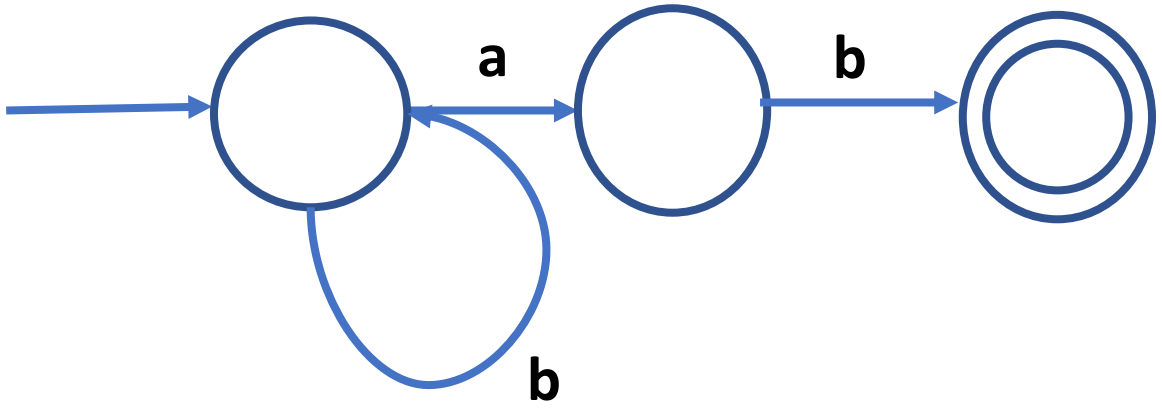
Automate fini complet ou incomplet ?

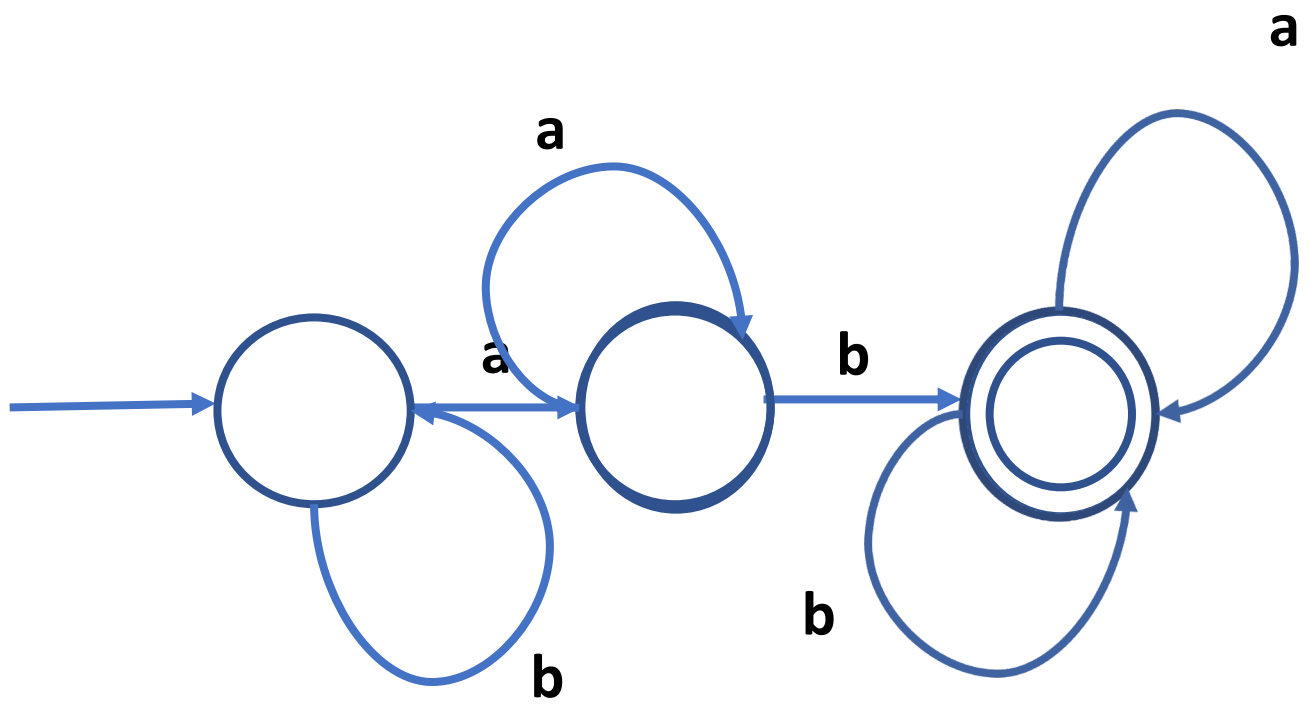


Compléter l’automate

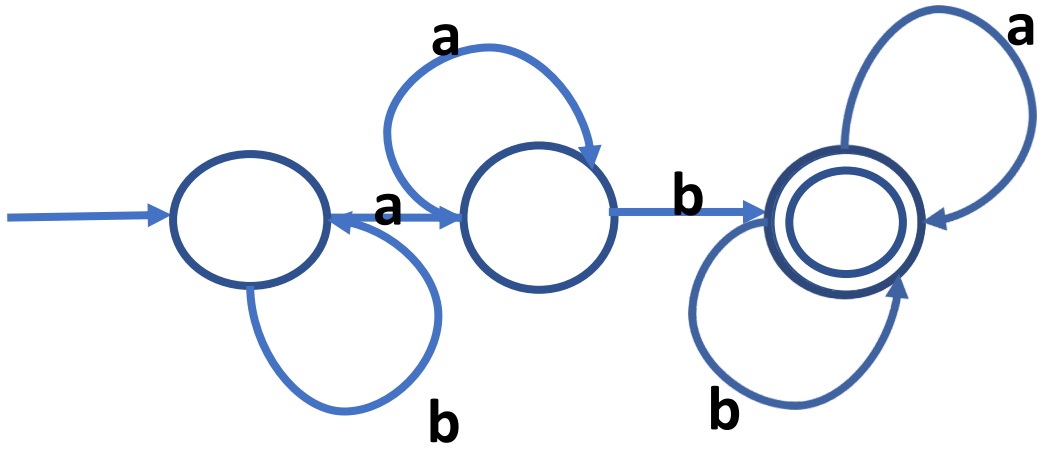


Automate fini complet

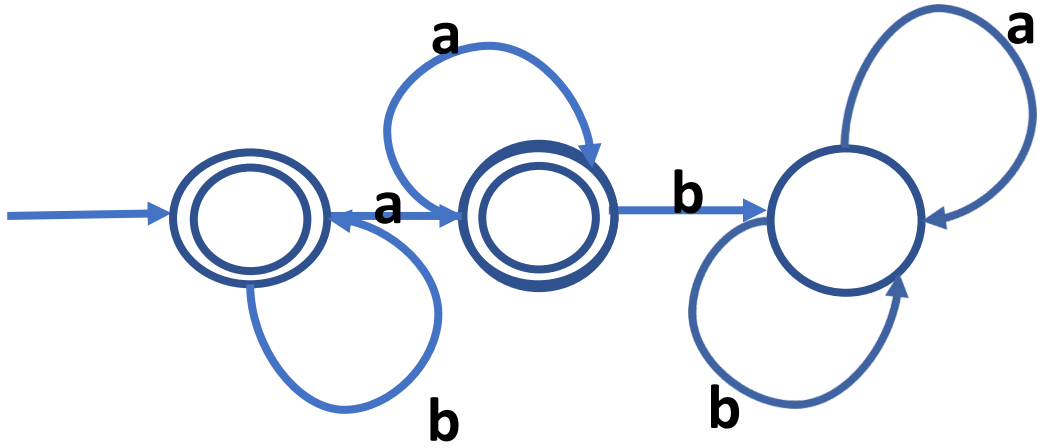




Négation : on construit un automate complet, puis on inverse les états terminaux



Cet automate reconnaît tous les mots qui contiennent l'affixe « ab »



Cet automate reconnaît tous les mots qui ne contiennent pas l'affixe « ab »

Application des automates finis

- Représenter les entrées d'un dictionnaire de mots simples et de mots composés sous la forme d'un automate fini :

table

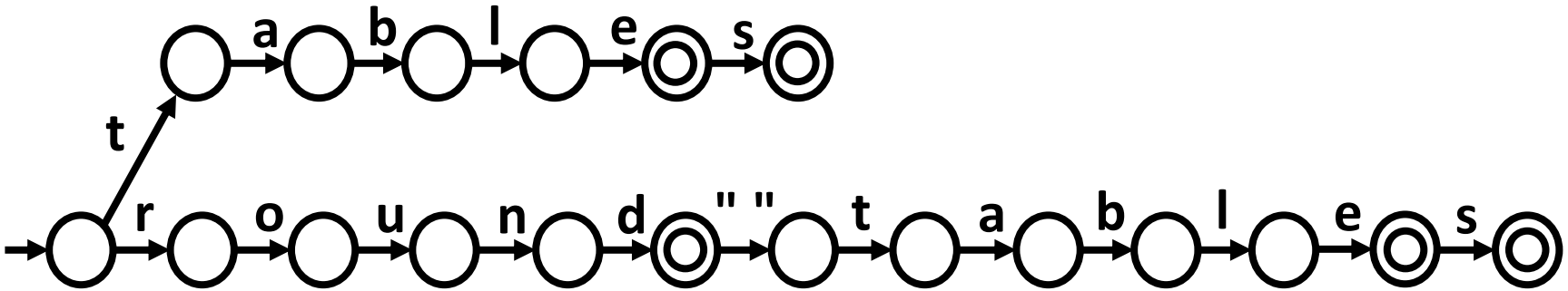
round

round table

- L'automate doit reconnaître toutes les formes : *table*, *tables*, *round table*, *round tables*.

Application des automates finis

- Représenter les entrées d'un dictionnaire de mots simples et de mots composés sous la forme d'un automate fini :



Automates finis avec sorties (*outputs*) : transducteurs finis

- Représenter les entrées d'un dictionnaire de mots simples et de mots composés sous la forme d'un transducteur qui produit leur analyse

table,N+Conc

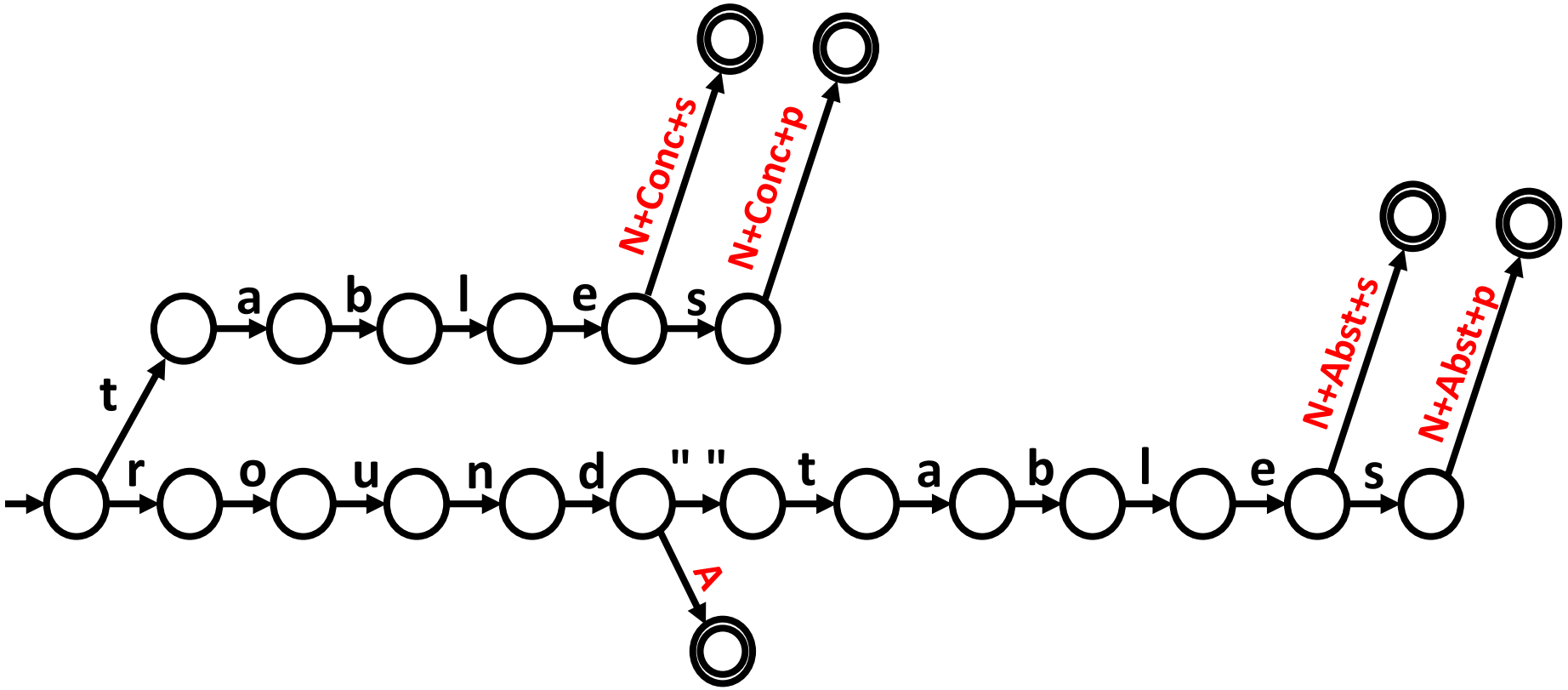
rond,A

table ronde,N+Abst

- L'automate doit reconnaître toutes les formes : *table*, *tables*, *round table*, *rond tables* et produire leurs propriétés lexicales (N, Conc, A, Abst) et aussi leur propriétés morphologiques +s ou +p

Application des automates finis

- Représenter les entrées d'un dictionnaire de mots simples et de mots composés



Exercice

- Représenter les entrées d'un dictionnaire sous la forme d'un transducteur déterministe

avant,ADVERBE

aviateur,NOM

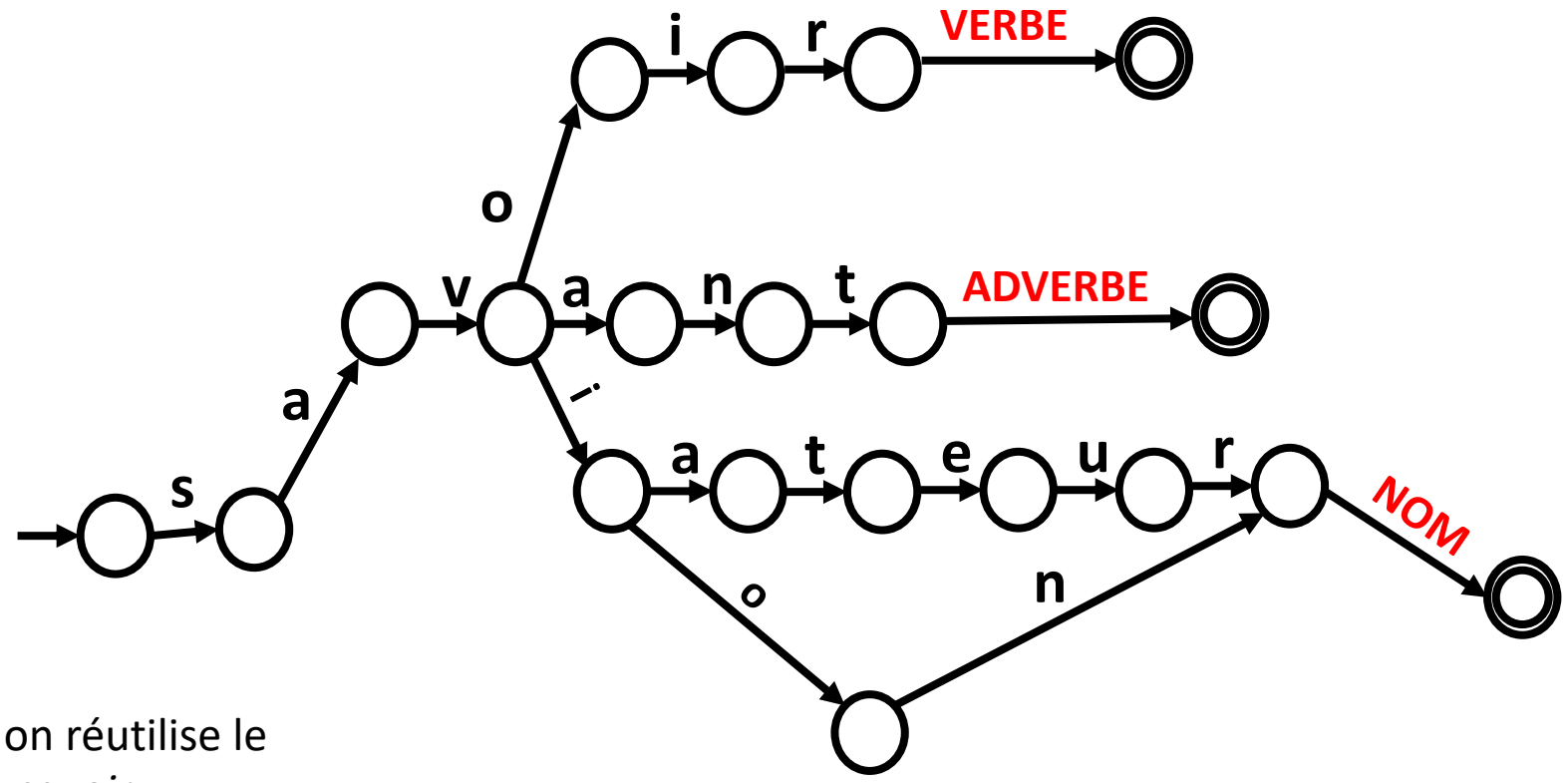
avoir,VERBE

avion,NOM

savoir,VERBE

Application des automates finis

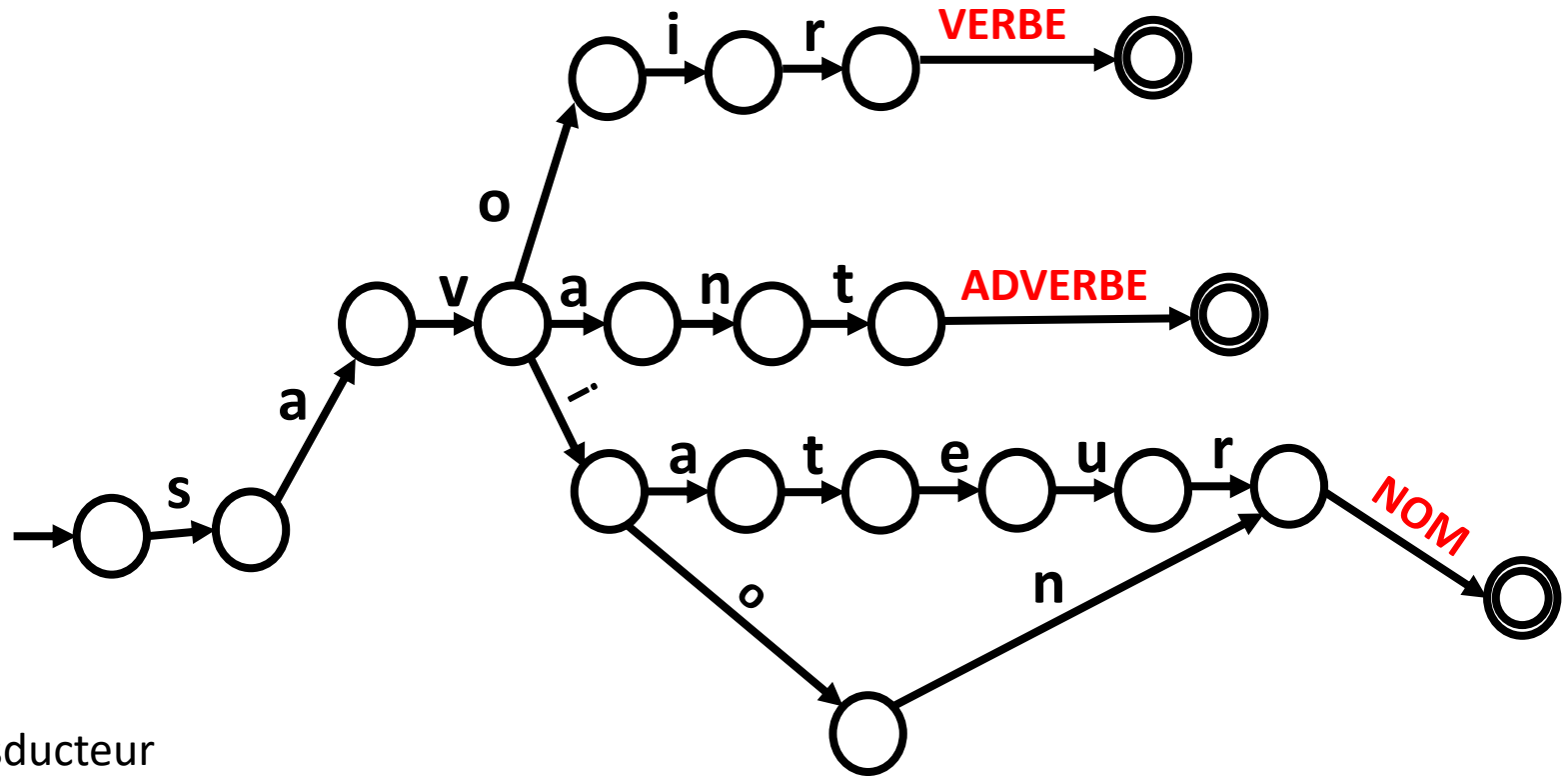
- Représenter les entrées d'un dictionnaire sous la forme d'un transducteur déterministe



Première tentative : on réutilise le chemin pour *avoir* et *savoir*

Application des automates finis

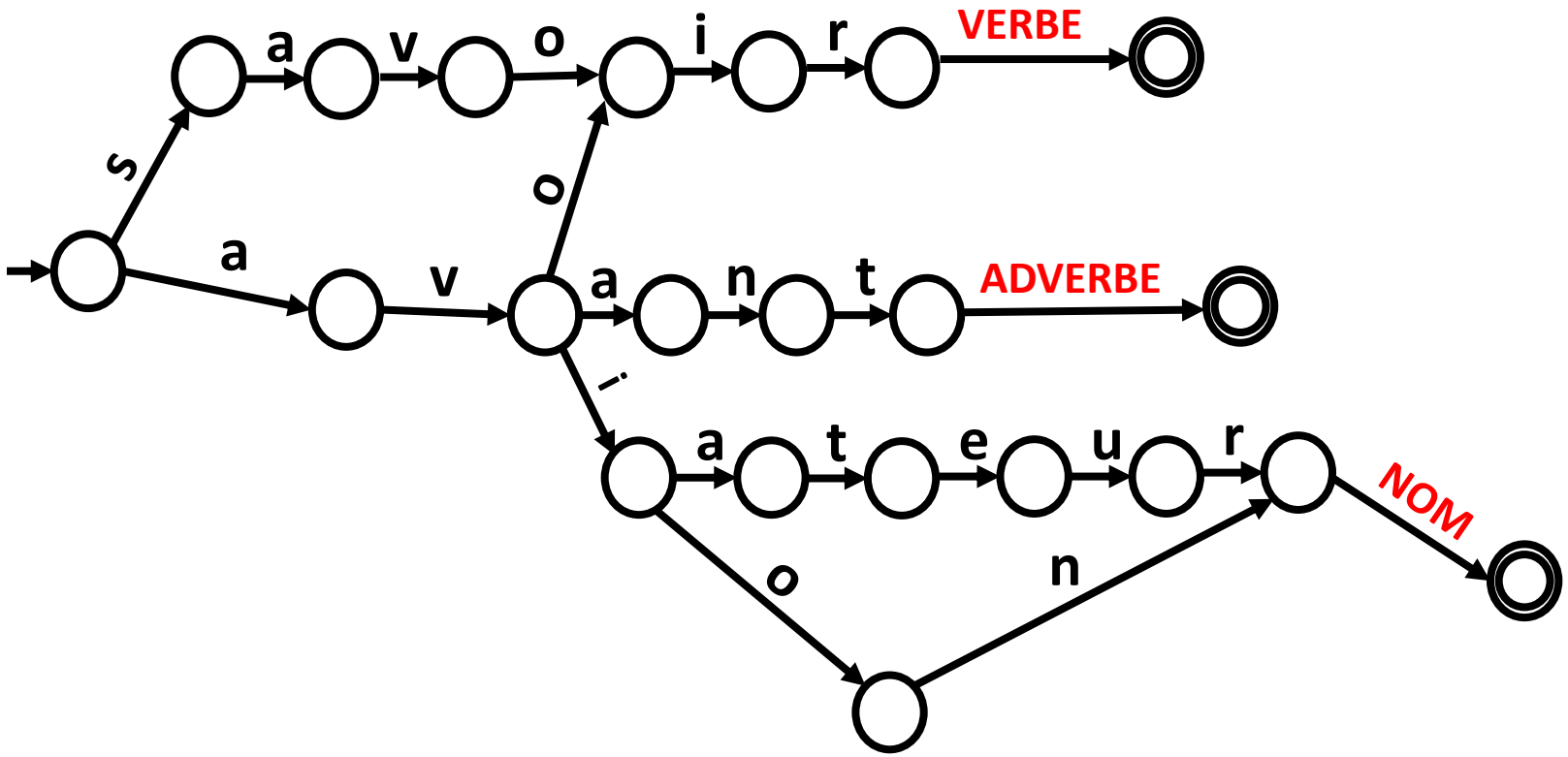
- Représenter les entrées d'un dictionnaire sous la forme d'un transducteur déterministe



PROBLÈME : ce transducteur reconnaît aussi *savant*, *savion* et *saviateur*

Application des automates finis

- Représenter les entrées d'un dictionnaire sous la forme d'un transducteur déterministe



Pour en savoir plus sur les machines

- Grammaires régulières ⟶ Automates finis

  https://fr.wikipedia.org/wiki/Automate_fini
- Grammaires hors-contextes ⟶ Automates à pile

  https://fr.wikipedia.org/wiki/Automate_à_pile
- Grammaires contextuelles ⟶ Automates linéairement bornés

  https://fr.wikipedia.org/wiki/Automate_linéairement_borné
- Grammaires non-restreintes ⟶ Machines de Turing

  https://fr.wikipedia.org/wiki/Machine_de_Turing

Récapitulation

- Equivalence entre grammaires, langages et machines
- Les grammaires génératives sont un mécanisme mathématique pour décrire n'importe quel langue naturelle
- Hiérarchie de Chomsky-Schützenberger : 4 types de grammaires génératives
- Les grammaires génératives ne sont pas adaptées au développement de grammaires de grande taille ; on a donc inventé des formalismes plus adaptés pour décrire des
  - grammaires régulières (ou rationnelles), ex. XFST, HFST, NooJ
  - grammaires hors-contexte (ou algébriques), ex. YACC, GPSG, NooJ
  - grammaires contextuelles, ex. LFG, TAG, NooJ
  - grammaires non restreintes, ex. HPSG, NooJ
- Chaque type de grammaire peut être traité par des algorithmes plus ou moins efficaces ; plus la grammaire est puissante, plus elle est complexe, et moins l'algorithme est rapide.

Récapitulation : exercices à rendre

Sur l'alphabet { a, b, c }, construire la grammaire

qui représente le langage qui contient :

1. (1 pt) dont la longueur est un multiple de 2
2. (1 pt) qui ne contiennent que des « a », ou alors que des « b »
3. (1 pt) qui ont un nombre pair de « a »

Sur l'alphabet français, construire la grammaire

qui représente le langage qui contient :

4. (1 pt) tous les mots communs français qui contiennent un « ë »
5. (1 pt) toutes les formes conjuguées du verbe *voler*
6. (2 points) Sur le vocabulaire V = {il, donne, le, la, les, lui, leur, me, te, se, nous, vous, en, y}, construire la grammaire qui représente le langage qui contient toutes les séquences de mots correctes en français

Récapitulation : exercices à rendre

7. (1 pt) Construire la grammaire qui reconnaît toutes les phrases que l'on peut construire à partir du vocabulaire V = { Anne, voisine, donne, à, un, une, le, la, son, sa, cadeau, pomme}

8. (1 pt) Le mot anglais *that* a quatre fonctions potentielles : DETERMINANT, PRONOM, CONJONCTION ou ADVERBE. Sur le modèle de la grammaire p. 59, construire une grammaire pour le désambiguïser dans certains cas.

9. (2 pts) Construire la grammaire qui reconnait des dates en français :

Date = ...

Jour = (*lundi* | *mardi* | *mercredi* | *jeudi* | *vendredi* | *samedi* | *dimanche*)

Mois = (*janvier* | *février* | *mars* | *avril* | *mai* | *juin* | *juilllet* | *août* | *septembre* | *octobre* | *novembre* | *décembre*)

... = ...

On pourra utiliser le symbole auxiliaire Nombre2-29.

La grammaire doit reconnaître : *lundi* ; *mardi 30 avril* ; *le 2 mai* ; le 31, mais ne doit pas reconnaître : ~~*vendredi 31 février*~~ ; ~~*samedi juin*~~ ; ~~27 mai~~ ; ~~*1 juillet*~~

Récapitulation : exercices à rendre

10. (2 pts) Construire une grammaire qui reconnaît toutes les phrases qui contiennent une négation, une opinion ou un sentiment négatif, ex. :

Stéphane est un looser ; Marie n'est pas une gagnante

Stupidement, Jeanne est repartie ; Eric a peur de son ombre

On pourra utiliser les symboles auxiliaires NomPos et NomNeg. On donnera quelques exemples de verbes et adverbes positifs ou négatifs, ex. *stupidement*, *intelligemment*, *gagner*, *perdre*

Récapitulation : exercices à rendre

11. (1 pt) Sur l'alphabet {a, b}, construire l'automate déterministe qui reconnaît tous les mots qui se terminent par « ab »
12. (1 pt) Sur l'alphabet {a, b, c}, construire l'automate déterministe pour l'expression :

((a | b)* c)*

Sur l'alphabet français :

13. (1 pt) Construire l'automate déterministe qui reconnaît toutes les variantes orthographiques et morphologiques du nom *tsar* (cf. exercice corrigé en cours)

14. (2 pts) Compléter la grammaire des noms de personne (cf. exercice corrigé en cours) pour reconnaître des séquences qui contiennent des titres et leur abréviation (ex. *Général*, *Monseigneur*, *Docteur*, etc.)

15. (1 pt) Construire l'automate déterministe qui reconnaît toutes les formes dérivées du verbe *monter* qui sont des lemmes (non fléchis)