Notes de cours

Langages formels

Frédéric Blanqui (INRIA)

version du 24/10/18

Retrouvez les informations sur le cours, les notes de cours et les TDs sur: `http://rewriting.gforge.inria.fr/lsf.html`.

Pour plus de détails sur certains aspects, voir par exemple:

- "Langages formels, calculabilité et complexité", Olivier Carton, Editions Vuibert, ISBN : 978-2-311-01400-6.
- "The theory of parsing, translation, and computing. Volume I: Parsing", Alfred Aho et Jeffrey Ullman, Prentice-Hall.

Contents

Cours 1 (3h30)

Dans ce cours, nous nous intéressons à différentes manières de définir des langages pour représenter des données (e.g. XML) ou des programmes (e.g. C, Java, OCaml), et comment vérifier qu'un fichier est bien un fichier XML ou un programme C, Java, etc. sans erreurs syntaxiques.

Nous verrons plusieurs manières de définir des langages: avec une grammaire (nous ne considérons que des grammaires dites algébriques ou hors-contextes), des "machines" (automate ou automate à pile), ou une "expression régulière".

Dans le 1er cours, nous introduirons la notion de grammaire algébrique et établirons un certain nombre de propriétés fondamentales des grammaires algébriques et de la classe des langages définissables par une grammaire algébrique. Nous verrons également un certain nombre de transformations permettant de simplifier la présentation d'une grammaire ou de la mettre sous une forme particulière.

Dans le 2e cours, nous introduirons la notion d'automate. Nous verrons que l'analyse syntaxique d'un langage défini par une forme très simple de grammaire dite linéaire peut être implémentée de manière très efficace en utilisant un automate (cela fera l'objet d'un TP).

Dans le 3e cours, nous verrons comment minimiser le nombre d'états d'un automate. Nous verrons également que les automates avec ε-transitions ne sont pas plus puissants que les automates standards, et quels sont les propriétés de la classe des langages reconnaissables par automate.

Dans le 4e cours, nous introduirons la notion d'expression régulière. Nous verrons alors que grammaires linéaires, automates et expressions régulières sont en fait équivalents et définissent la même classe de languages, mais que ces trois techniques sont limitées: elles ne permettent pas de traiter les langages bien parenthésés.

C'est ainsi que, dans le 5e et dernier cours, nous considérons la classe des langages dits $LL(k)$ qui peuvent être analysés efficacement en utilisant un automate à pile.

1 Mots et langages

Un *alphabet* Σ est un ensemble fini quelconque dont les éléments sont appelés *lettres*. Par exemple, $\Sigma = \{a, b, c, \ldots, 0, 1, \ldots, +, *, (, \ldots\}$.

Un *mot* u est une suite finie de lettres juxtaposées (un fichier XML ou C peut être vu comme un mot sur l'alphabet des caractères ASCII). Par exemple, aac est un mot sur $\Sigma = \{a, b, c\}$. La longueur d'un mot est le nombre de lettres qu'il contient. Par exemple, aac est de longueur 3. L'unique mot de longueur nulle est noté ε et appelé le *mot vide*. L'ensemble des mots sur Σ est noté Σ^*.

La concaténation d'un mot $a_1 \dots a_p$ et d'un mot $b_1 \dots b_q$ est le mot $a_1 \dots a_p b_1 \dots b_q$.

Remarque: La concaténation est une opération *associative* ($(ab)c = a(bc)$ pour tout a, b, c) ayant ε comme *élément neutre* ($a\varepsilon = \varepsilon a = a$ pour tout a). Muni de la concaténation, Σ^* est ainsi un *monoïde*.

On peut itérer la concaténation d'un mot: u répété n fois est noté u^n. Par récurrence, $u^0 = \varepsilon$ et $u^{n+1} = uu^n$.

Un mot u est un facteur d'un mot v si v est de la forme aub avec $a, b \in \Sigma^*$. C'est un préfixe (resp. suffixe) si, de plus, $a = \varepsilon$ (resp. $b = \varepsilon$).

Un *language* est un ensemble de mots, c'est-à-dire, une partie de Σ^*.

On peut étendre les opérations sur les mots aux langages. Si L_1 et L_2 sont deux langages sur le même alphabet Σ, alors $L_1 L_2 = \{u_1 u_2 \in \Sigma^* | u_1 \in L_1, u_2 \in L_2\}$. Si L est un langage, soit L^n le langage des puissances de n d'un mot de L: $L^0 = \{\varepsilon\}$ et $L^{n+1} = LL^n$. On note le langage de toutes les puissances possibles d'un mot de L par:

$$L^* = \bigcup_{n \in \mathbb{N}} L^n.$$

C'est le plus petit langage clos par concaténation et contenant L.

2 Grammaires

Une manière de définir un language est d'utiliser une grammaire.

Une *grammaire* G est un quadruplet $(\Sigma, \mathcal{N}, \mathcal{R}, S)$ où Σ est un alphabet (fini) de *terminaux*, $\mathcal{N}$ est un alphabet (fini) de *non-terminaux* contenant S (non-terminal de départ), et $\mathcal{R}$ est un ensemble fini de régles $l \to r$ où $l, r \in (\Sigma \cup \mathcal{N})^*$.

Une règle de la forme $X \to \varepsilon$ est dite une ε-règle. Une règle de la forme $X \to Y$ est dite *unaire*.

Un mot u *se réécrit* (en une étape) en v, noté $u \to_G v$ (ou simplement $u \to v$ s'il est clair de quelle grammaire il s'agit), si u est de la forme alb avec $a, b \in \Sigma^*$, v est de la forme arb et $l \to r \in \mathcal{R}$.

On note par $u \to^* v$ si u se réécrit en v en $n \geq 0$ étapes. On dit alors que u *se réduit en* v. Par exemple, avec $\mathcal{R} = \{S \to 1, S \to 2, S \to S + S, S \to S * S\}$, noté plus simplement $\{S \to 1 \mid 2 \mid S + S \mid S * S\}$, nous avons la réduction suivante (en soulignant le non-terminal réécrit à chaque étape):

$$\underline{S} \to \underline{S} + S \to 1 + \underline{S} \to 1 + \underline{S} * S \to 1 + 2 * \underline{S} \to 1 + 2 * 2 \qquad (1)$$

La relation $\to^*$ est la clôture réflexive et transitive de $\to$, c'est-à-dire, la plus petite relation reflexive et transitive contenant $\to$. Si on définit $\to^0$ comme la

relation égalité et $\to^n$ la réécriture en n étape ($\to^{n+1} = \to\to^n$), alors

$$\to^* = \bigcup_{n \geq 0} \to^n.$$

On note par $\to^+ = \bigcup_{n \geq 1} \to^n$ la clôture transitive de $\to$.

Le *langage engendré* par G *à partir d'un mot* $u \in (\Sigma \cup \mathcal{N})^*$ est l'ensemble des mots de Σ^* vers lesquels u peut se réécrire:

$$L_G(u) = \{v \in \Sigma \mid u \to^* v\}.$$

Nous définissons alors le *langage engendré par* G, $L(G)$, comme étant $L_G(S)$.

Il peut exister de nombreuses grammaires différentes pour un même langage. Deux grammaires G_1 et G_2 sont dites *équivalentes* si elles engendrent le même langage.

Deux grammaires équivalentes peuvent avoir des propriétés très différentes. On aura donc parfois intérêt à transformer une grammaire en une autre grammaire ayant de meilleures propriétés.

Exemples de grammaires:

- Grammaire Yacc d'ANSI C:
  `https://www.lysator.liu.se/c/ANSI-C-grammar-y.html`
- Grammaire de C11 (Annex A, p. 476):
  `http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf`
- Grammaire de Python 3.7:
  `https://docs.python.org/3/reference/grammar.html`
- Grammaire d'OCaml 4.07:
  `https://caml.inria.fr/pub/docs/manual-ocaml-4.07/language.html`
- Liste de grammaires ANTLR:
  `https://github.com/antlr/grammars-v4`

3 Grammaires algébriques

Une grammaire est dite *algébrique* (ou hors-contexte ou encore de type 2) si toutes les règles sont de la forme $X \to u$ avec $X \in \mathcal{N}$. Dans la suite, nous ne considérons que des grammaires algébriques.

Un langage L est algébrique s'il existe une grammaire algébrique G telle que $L = L(G)$.

Pour toute grammaire (y compris les grammaires non algébriques), si $a \to^p a'$ et $b \to^q b'$, alors $ab \to^{p+q} a'b'$. Cette propriété peut-être inversée quand la grammaire est algébrique:

Lemme 1 Avec une grammaire algébrique, si $ab \to^n c$, alors il existe a', b', p, q tels que $c = a'b'$, $n = p + q$, $a \to^p a'$ et $b \to^q b'$.

Preuve. Par récurrence sur n. Le cas $n = 0$ est immédiat. Si $ab \to c$ alors, comme la grammaire est algébrique, il existe α, X, β, r tels que $ab = \alpha X \beta$ and $c = \alpha r \beta$. Soit le X est dans a, c'est-à-dire, $\beta = \alpha' b$, et il suffit de prendre $p = 1$, $a' = \alpha r \alpha'$, $q = 0$ et $b' = b$. Soit le X est dans b, c'est-à-dire, $\alpha = a\beta'$, et il suffit de prendre $p = 0$, $a' = a$, $q = 1$ et $b' = \beta' r \beta$. Ainsi, le cas $n \geq 1$ suit facilement par hypothèse de récurrence. ■

Avec une grammaire algébrique, l'ensemble des mots que l'on peut engendrer à partir de ab est la concaténation de l'ensemble des mots que l'on peut engendrer à partir de a et de l'ensemble des mots que l'on peut engendrer à partir de b, qui sont indépendants l'un de l'autre:

Corollaire 1 Si G est une grammaire algébrique, alors $L_G(ab) = L_G(a)L_G(b)$.

De plus, la longueur de la réduction pour aller de a à a', ou de b à b', n'est pas plus grande que celle pour aller de ab à $a'b'$ (c'est même une sous-réduction), ce qui peut être utile quand on raisonne par récurrence sur la longueur des réductions.

Remarquons enfin que l'ordre de réduction des non-terminaux n'a pas d'importance. Ainsi, dans les grammaires algébriques, on peut se limiter à une stratégie particulière de dérivation, par exemple $\to_g$ (respectivement $\to_d$) la relation consistant à réécrire le non-terminal le plus à gauche (respectivement à droite). Les langages reconnus en utilisant $\to$, $\to_g$ ou $\to_d$ sont identiques:

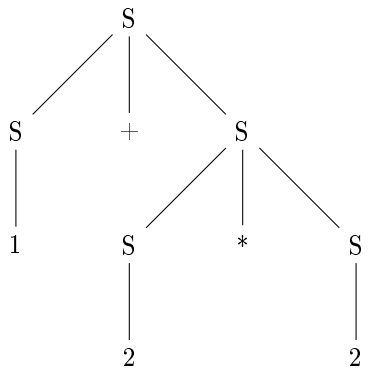
Lemme 2 Si G est une grammaire algébrique, alors $L(G) = L_g(G) = L_d(G)$ où $L_g(G) = \{u \in \Sigma^* \mid S \to_g^* u\}$ et $L_d(G) = \{u \in \Sigma^* \mid S \to_d^* u\}$.

Preuve. Détaillons le cas de $L_g(G)$. La preuve pour $L_d(G)$ est similaire. Nous avons $L_g(G) \subseteq L(G)$ puisque $\to_g \subseteq \to$. Montrons maintenant que, si $X \in \Sigma \cup \mathcal{N}$ et $X \to^n u \in \Sigma^*$, alors $X \to_g^n u$, par récurrence sur n. Si $n = 0$, c'est immédiat. Supposons alors $n > 0$. Alors, $X \to_g x_1 \dots x_k \to^{n-1} u$. Ainsi, il existe $u_1, \dots, u_k$ et $n_1, \dots, n_k$ tels que $u = u_1 \dots u_k$, $x_i \to^{n_i} u_i$ et $n - 1 = \Sigma_{i=1}^k n_i$. Par hypothèse de récurrence, $x_i \to_g^{n_i} u_i$. Donc, $X \to_g^n u$. ■

3.1 Arbres de dérivation

Un *arbre de dérivation* est un arbre dont les noeuds sont étiquettés par un élément de $\Sigma \cup \mathcal{N}$, dont la racine est étiquetée par S et tel que, si un noeud est étiqueté par X et ses sous-arbres sont étiquetés par $a_1, \dots, a_n$ respectivement, alors $X \to a_1 \dots a_n \in \mathcal{R}$.

A toute séquence de réécriture aboutissant à un mot u, on peut associer un arbre de dérivation dont les feuilles, prises de gauche à droite, forment le mot u. Par exemple, la réduction (1) a l'arbre de dérivation:



Inversement, à tout arbre de dérivation, on peut associer une séquence de réécritures aboutissant au mot formé par les feuilles de l'arbre prises de gauche à droite. Par exemple, on réécrivant à chaque étape le non-terminal le plus à gauche possible, comme dans la réduction (1).

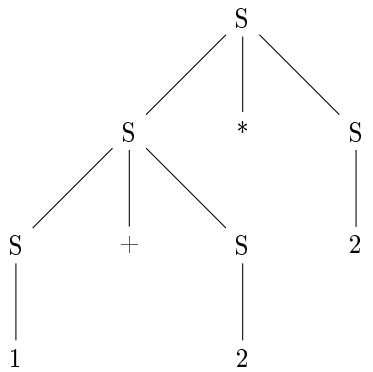
Dans une grammaire quelconque, le résultat d'une réduction dépend du choix, à chaque étape, de la règle utilisée et de la position à laquelle on l'applique. Dans une grammaire algébrique, le résultat ne dépend que du choix de la règle: deux réductions qui ne diffèrent que par le choix des positions ont le même arbre de dérivation. Ainsi, la réduction

$$\underline{S} \to \underline{S} + S \to 1 + \underline{S} \to 1 + S * \underline{S} \to 1 + \underline{S} * 2 \to 1 + 2 * 2$$

(le 2e argument de $*$ est réduit avant son 1er argument) a le même arbre de dérivation que la réduction (1). Par contre,

$$\underline{S} \to S * \underline{S} \to \underline{S} * 2 \to \underline{S} + S * 2 \to 1 + \underline{S} * 2 \to 1 + 2 * 2$$

(on applique $S \to S * S$ avant $S \to S + S$) a un arbre de dérivation différent même si le mot engendré est le même $(1 + 2 * 2)$:



On dit qu'une grammaire G est *ambiguë* s'il existe un mot $u \in L(G)$ ayant deux arbres de dérivations différents. Ainsi, la grammaire de (1) est ambiguë:

$1 + 2 * 2$ peut être interprété soit comme $1 + (2 * 2) = 5$ (c'est le premier arbre de dérivation), soit comme $(1 + 2) * 2 = 6$ (c'est le second arbre de dérivation).

Remarque: il est indécidable de savoir si une grammaire est ambiguë ou pas. Certains langages algébriques sont inhéremment ambiguë: il n'existe aucune grammaire non-ambiguë les générant.

3.2 Résolution d'équations par itération

Nous allons commencer par voir un théorème très utile pour justifier la terminaison d'un algorithme consistant à itérer une fonction que nous utiliserons de nombreuses fois par la suite.

Etant donné un ensemble E, soit $\wp(E)$ l'ensemble des parties de E.

Une relation $\leq$ sur E est une relation d'ordre si elle est réflexive, transitive et antisymétrique (si $x \leq y$ et $y \leq x$, alors $x = y$).

Une fonction F sur E est monotone si $x \leq y$ implique $F(x) \leq F(y)$.

Un élément x est un minorant (resp. majorant) d'une partie A de E si x est plus petit (resp. grand) que tout élément de A.

Un élément x est une borne inférieure (resp. supérieure) d'une partie A de E si x est un minorant (resp. majorant) de A et tout minorant (resp. majorant) de A est plus grand que x.

Un ensemble E muni d'une relation d'ordre $\leq$ est un treillis complet si toute partie de E admet une borne inférieure et une borne supérieure.

Par exemple, pour tout ensemble $\mathcal{N}$, $(\wp(\mathcal{N}), \subseteq)$ est un treillis: la borne supérieure d'une famille $(A_i)_{i \in I}$ de parties de $\mathcal{N}$ est $\bigcup_{i \in I} A_i$, la borne supérieure de $\emptyset$ est $\emptyset$, et la borne supérieure d'une famille non vide $(A_i)_{i \in I}$ est $\bigcap_{i \in I} A_i$.

Autre exemple de treillis: étant donné un ensemble quelconque D et un treillis $(E, \leq_E)$, l'ensemble $D \to E$ des fonctions de D dans E avec l'ordre $f \leq g$ ssi, pour tout $x \in D$, $f(x) \leq_E f(y)$.

Théorème 1 (Tarski) Si $(E, \leq)$ est un treillis complet à n éléments et F une fonction monotone sur E, alors l'équation $F(\mathcal{Y}) = \mathcal{Y}$ admet une plus petite solution A qu'on peut calculer de la façon suivante. Soit A_0 le plus petit élément de E et, pour tout $i \in \mathbb{N}$, soit $A_{i+1} = F(A_i)$. Alors, il existe $k \leq n$ tel que $A = A_k$.

Preuve. Montrons que, pour tout $i \in \mathbb{N}$, $A_i \leq A_{i+1}$, par récurrence sur i. Si $i = 0$, nous avons $A_i = A_0 \leq A_1$, quelque soit A_1. Si $i > 0$, alors nous avons $A_{i-1} \leq A_i$ par hypothèse de récurrence. Donc, par monotonie, nous avons $A_i = F(A_{i-1}) \leq F(A_i) = A_{i+1}$.

Maintenant, comme E est fini, il y a un nombre fini d'inclusions strictes. Donc, il existe $k \leq n$ tel que, pour tout $i \geq k$, $A_i = A_k$. En particulier, A_k est solution de $F(\mathcal{Y}) = \mathcal{Y}$.

Montrons maintenant que c'est la plus petite solution. Soit $\mathcal{Y}$ une partie de E telle que $F(\mathcal{Y}) = \mathcal{Y}$. Montrons que, pour tout $i \in \mathbb{N}$, $A_i \leq \mathcal{Y}$, par récurrence sur i. Si $i = 0$, alors $A_i = \emptyset \leq \mathcal{Y}$ quelque soit $\mathcal{Y}$. Si $i > 0$ alors, par hypothèse

de récurrence, $A_{i-1} \leq \mathcal{Y}$. Donc, par monotonie, $A_i = F(A_{i-1}) \leq F(\mathcal{Y}) = \mathcal{Y}$. ■

Ainsi, si E est un treillis fini et F est monotone, alors le programme suivant termine avec $X = Y = A$:

```
X := plus_petit_element; Y := F(X);
tant que X <> Y faire
  { X := Y; Y := F(X) }
```

3.3 Grammaires propres

Une grammaire algébrique $G = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ est *propre* si elle ne contient aucune règle unaire et, soit $\varepsilon \notin L(G)$ et G ne contient aucune ε-règle, soit $\varepsilon \in L(G)$, $S \to \varepsilon$ est la seule ε-règle de $\mathcal{R}$ et aucun membre droit de règle ne contient S.

Lemme 3 Si G est une grammaire algébrique, alors il existe une grammaire algébrique G' ne contenant pas ε et telle que $L(G') = L(G) - \{\varepsilon\}$.

Preuve. Soit $G = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ une grammaire algébrique.

Première étape: on calcule l'ensemble $\mathcal{E}$ des non-terminaux X effaçables, c'est-à-dire, tel que $X \to_G^* \varepsilon$.

On remarque que $\mathcal{E} = F(\mathcal{E})$ où F est la fonction sur $\wp(\mathcal{N})$ telle que $F(\mathcal{Y}) = \{X \in \mathcal{N} \mid \exists r \in \mathcal{Y}^*, X \to r \in \mathcal{R}\}$. En effet, si $X \to^* \varepsilon$ alors il existe $X \to r \in \mathcal{R}$ tel que $X \to r \to^* \varepsilon$. Comme $r \to^* \varepsilon$, $r = X_1 \dots X_n$ avec $X_i \in \mathcal{N}$. Donc, pour tout i, $X_i \in \mathcal{E}$ et $X \in F(\mathcal{E})$. Réciproquement, si $X \in F(\mathcal{E})$ alors il existe $X \to r \in \mathcal{R}$ tel que $r \in \mathcal{E}^*$. Donc, $X \to^* \varepsilon$.

On vérifie ensuite que F est monotone: si $\mathcal{Y} \subseteq \mathcal{Z}$, alors $F(\mathcal{Y}) \subseteq F(\mathcal{Z})$. En effet, soit $X \in F(\mathcal{Y})$. Alors, il existe $r \in \mathcal{Y}^*$ tel que $X \to r \in \mathcal{R}$. Comme $\mathcal{Y}^* \subseteq \mathcal{Z}^*$, nous avons $X \in F(\mathcal{Z})$.

Ainsi, d'après le théorème de Tarski, l'équation $F(\mathcal{Y}) = \mathcal{Y}$ admet une plus petite solution A calculable par itération de F à partir de $\emptyset$. Donc, $A \subseteq \mathcal{E}$.

On prouve enfin que $\mathcal{E} \subseteq A$ en montrant que si $X \to^k \varepsilon$ alors $X \in A$, par récurrence sur $k \geq 1$. Si $k = 1$, alors $X \to \varepsilon \in \mathcal{R}$ et $X \in F(\emptyset) \subseteq A$. Si $k > 1$, alors $X \to X_1 \dots X_n$ et, pour tout i, $X_i \to^{n_i} \varepsilon$ avec $n_i < n$. Par hypothèse de récurrence, $X_i \in A$. Donc, $X \in F(A) = A$.

Deuxième étape: pour chaque règle $X \to u \in \mathcal{R}$, rajouter toutes les règles $X \to v$ où v est obtenu en remplaçant dans u un nombre quelconque de non-terminaux effaçables par ε, et éliminer toutes les règles de la forme $X \to \varepsilon$. On obtient ainsi $G' = (\Sigma, \mathcal{N}, \mathcal{R}', S)$ où $\mathcal{R}' = \{X \to v \mid X \to u \in \mathcal{R}, u \to_{\mathcal{E}}^* v, v \neq \varepsilon\}$ où $\to_{\mathcal{E}}$ est la relation de réécriture engendrée par les règles $\{X \to \varepsilon \mid X \in \mathcal{E}\}$.

Montrons maintenant que $L(G') \subseteq L(G) - \{\varepsilon\}$. Soit $u \in L(G')$. Alors $S \to_{G'}^* u$. Par définition, nous avons $\to_{G'} \subseteq \to_G \to_{\mathcal{E}}^*$ et $\to_{\mathcal{E}} \subseteq \to_G^*$. Donc, $S \to_G^* u$.

Montrons enfin que $L(G) - \{\varepsilon\} \subseteq L(G')$. Soit $u \in L(G) - \{\varepsilon\}$. Alors $S \to_G^* u$ et $u \neq \varepsilon$. Montrons que $S \to_{G'}^* u$ par récurrence sur le nombre de ε-règles utilisées, c'est-à-dire, de la forme $X \to \varepsilon$. Si aucune ε-règle n'est utilisée, alors

$S \rightarrow^*_{G'} u$ car toute règle de G qui n'est pas de la forme $X \rightarrow \varepsilon$ est incluse dans G'. Si une ε-règle est utilisée, nous avons une étape de réécriture de la forme $aXb \rightarrow_G ab$. Celle-ci ne peut pas être la première étape de réécriture car, sinon, nous aurions $X = S$ et $ab = \varepsilon = u$. Donc celle-ci est précédée d'une séquence de réécritures de la forme $cYd \rightarrow_G c\alpha X\beta d \rightarrow^*_G aXb$ avec $Y \rightarrow \alpha X\beta \in \mathcal{R}$, $c\alpha \rightarrow^*_G a$ et $\beta d \rightarrow^*_G b$. On peut alors remplacer la séquence $cYd \rightarrow^*_G ab$ par $cYd \rightarrow_{G'} c\alpha\beta d \rightarrow^*_G ab$ qui utilise une ε-règle de moins. ■

Lemme 4 Toute grammaire algébrique G ne contenant pas ε est effectivement équivalente à une grammaire algébrique ne contenant pas ε et n'ayant pas de règle unaire.

Preuve. Soit $G = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ une grammaire algébrique.

Premièrement, pour tous non-terminaux X et Y tel que $X \rightarrow^* Y$, et toute règle $Y \rightarrow r \in \mathcal{R}$ avec $r \notin \mathcal{N}$, on rajoute la règle $X \rightarrow r$. Deuxièmement, on élimine toutes les règles unaires. On obtient ainsi $G' = (\Sigma, \mathcal{N}, \mathcal{R}', S)$ où $\mathcal{R}' = \{X \rightarrow u \mid X \rightarrow^*_G Y, Y \rightarrow u \in \mathcal{R}, u \notin \mathcal{N}\}$.

Etant donné X, comment calculer $\mathcal{R}^Z = \{X \in \mathcal{N} \mid Z \rightarrow^* X\}$? On remarque que $\mathcal{R}^Z = F(\mathcal{R}^Z)$ où F est la fonction sur $\wp(\mathcal{N})$ telle que $F(\mathcal{Y}) = \{Z\} \cup \{X \in \mathcal{N} \mid \exists Y \in \mathcal{Y}, Y \rightarrow X \in \mathcal{R}\}$. En effet, supposons que $Z \rightarrow^k X$. Si $k = 0$ alors $Z = X$. Donc, $X \in F(\mathcal{R}^Z)$. Sinon, comme G ne contient pas ε, il existe $Y \rightarrow X \in \mathcal{R}$ tel que $Z \rightarrow^* Y \rightarrow X$. Donc, $X \in F(\mathcal{R}^Z)$. Réciproquement, supposons que $X \in F(\mathcal{R}^Z)$. Si $X = Z$ alors $X \in \mathcal{R}^Z$. Sinon il existe $Y \in \mathcal{R}^Z$ tel que $Y \rightarrow X \in \mathcal{R}$. Donc, $X \in \mathcal{R}^Z$.

On vérifie ensuite que F est monotone: si $\mathcal{Y} \subseteq \mathcal{Z}$, alors $F(\mathcal{Y}) \subseteq F(\mathcal{Z})$. Ainsi, d'après le théorème de Tarski, l'équation $F(\mathcal{Y}) = \mathcal{Y}$ admet une plus petite solution A calculable par itération de F à partir de $\emptyset$. Donc, $A \subseteq \mathcal{R}^Z$. Prouvons maintenant que $\mathcal{R}^Z \subseteq A$ en montrant que si $Z \rightarrow^k X$ alors $X \in A$, par récurrence sur k. Si $k = 0$, alors $X = Z \in F(\emptyset) \subseteq A$. Si $k > 0$ alors, comme G ne contient pas ε, il existe $Y \rightarrow X \in \mathcal{R}$ tel que $Z \rightarrow^{k-1} Y \rightarrow X$. Par hypothèse de récurrence, $Y \in A$. Donc, $X \in F(A) = A$.

Nous avons $L(G') \subseteq L(G)$ car $\rightarrow_{G'} \subseteq \rightarrow^+_G$. Montrons maintenant que, si $S \rightarrow^*_G u$, alors $S \rightarrow^*_{G'} u$, par récurrence sur le nombre de règles unaires utilisées. Si une règle unaire est utilisée, alors la séquence de réécriture contient une sous-séquence de la forme $aXb \rightarrow_G aYb \rightarrow^*_G a'Yb' \rightarrow_G a'vb'$ avec $v \notin \mathcal{N}$. Mais cette sous-séquence peut être remplacée par $aXb \rightarrow_{G'} avb \rightarrow^*_G a'vb'$ qui utilise une règle unaire de moins. ■

Lemme 5 Toute grammaire algébrique est effectivement équivalente à une grammaire algébrique propre.

Preuve. Soit G une grammaire algébrique. Par le lemme 3, il existe une grammaire algébrique G_1 ne contenant pas ε et telle que $L(G_1) = L(G) - \{\varepsilon\}$. Par le lemme 4, il existe une grammaire algébrique $G_2 = (\Sigma, \mathcal{N}_2, \mathcal{R}_2, S_2)$ ne contenant pas ε et n'ayant pas de règle unaire telle que $L(G_2) = L(G_1)$. Si $\varepsilon \notin L(G)$, alors nous prenons G_2. Si par contre $\varepsilon \in L(G)$ alors nous prenons

$G_3 = (\Sigma, \mathcal{N}_3, \mathcal{R}_3, S_3)$ où S_3 est un nouveau non-terminal n'appartenant pas à $\mathcal{N}_2$, $\mathcal{N}_3 = \mathcal{N}_2 \cup \{S_3\}$ et $\mathcal{R}_3 = \mathcal{R}_2 \cup \{S_3 \to \varepsilon\} \cup \{S_3 \to u \mid S_2 \to u \in \mathcal{R}_2\}$. ■

3.4 Grammaires réduites

Un non-terminal X est *improductif* s'il ne génère aucun mot: $L_G(X) = \emptyset$. Il est *inaccessible* si aucun arbre de dérivation ne contient X. Il est *inutile* s'il est improductif ou inaccessible.

Une grammaire $G = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ est *réduite* si $\mathcal{R}$ ne contient aucun non-terminal inutile.

Lemme 6 Toute grammaire algébrique $G = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ est effectivement équivalente à une grammaire algébrique réduite $G' = (\Sigma, \mathcal{N}, \mathcal{R}', S)$ où $\mathcal{R}' \subseteq \mathcal{R}$.

Preuve. Nous procédons en deux étapes.

Première étape: on calcule l'ensemble $\mathcal{P}$ des non-terminaux productifs et on élimine toutes les règles contenant un non-terminal improductif. On obtient alors $G_1 = (\Sigma, \mathcal{N}, \mathcal{R}_1, S)$ où $\mathcal{R}_1 = \{X \to u \in \mathcal{R} \mid u \in (\Sigma \cup \mathcal{P})^*\}$.

Comment calculer $\mathcal{P}$? On remarque que $\mathcal{P} = F(\mathcal{P})$ où F est la fonction sur $\wp(\mathcal{N})$ telle que $F(\mathcal{Y}) = \{X \in \mathcal{N} \mid \exists X \to u \in \mathcal{R}, u \in (\Sigma \cup \mathcal{Y})^*\}$. On vérifie ensuite que F est monotone. Ainsi, d'après le théorème de Tarski, l'équation $\mathcal{Y} = F(\mathcal{Y})$ admet une plus petite solution $A \subseteq \mathcal{P}$ calculable par itération de F à partir de $\emptyset$. Nous prouvons maintenant que $\mathcal{P} \subseteq A$ en montrant que, si $X \to^k u \in \Sigma^*$ alors $X \in A$, par récurrence sur $k \geq 1$. Si $k = 1$ alors $X \in F(\emptyset) \subseteq A$. Si $k > 1$ alors il existe $X \to a_1 \dots a_n \in \mathcal{R}$ avec $a_i \in \Sigma \cup \mathcal{N}$, $u_1, \dots, u_n \in \Sigma^*$, et $k_1, \dots, k_n < n$ tels que $u = u_1 \dots u_n$ et, pour tout i, $a_i \to^{k_i} u_i \in \Sigma^*$. Si $a_i \in \mathcal{N}$ alors, par hypothèse de récurrence, $a_i \in A$. Donc, $X \in F(A) = A$.

Deuxième étape: à partir de G_1, on calcule l'ensemble $\mathcal{A}$ des non-terminaux accessibles depuis S et on élimine toutes les règles contenant un non-terminal inaccessible. On obtient alors $G_2 = (\Sigma, \mathcal{N}, \mathcal{R}_2, S)$ où $\mathcal{R}_2 = \{X \to u \in \mathcal{R}_1 \mid u \in (\Sigma \cup \mathcal{A})^*\}$.

Comment calculer $\mathcal{A}$? On remarque que $\mathcal{A} = G(\mathcal{A})$ où $G(\mathcal{Y}) = \{X \in \mathcal{N} \mid \exists u, \exists v, \exists Y \in \mathcal{Y}, Y \to uXv \in \mathcal{R}\}$. On vérifie que G est monotone. Ainsi, d'après le théorème de Tarski, l'équation $G(\mathcal{Y}) = \mathcal{Y}$ admet une plus petite solution $A \subseteq \mathcal{A}$ calculable par itération de G à partir de $\emptyset$. On prouve ensuite que $\mathcal{A} \subseteq A$ en montrant que, si $S \to^k uXv$ alors $X \in A$, par récurrence sur $k \geq 0$.

On vérifie maintenant que $L(G) = L(G_2)$. Puisque G et G_2 ont le même non-terminal de départ et $\mathcal{R}_2 \subseteq \mathcal{R}$, $L(G_2) \subseteq L(G)$. Montrons maintenant que $L(G) \subseteq L(G_2)$. Soit $u \in L(G)$. Alors $S \to_G^* u$. Si $X \to v$ est une règle utilisée dans cette dérivation, nous avons que X et tous les non-terminaux de v sont productifs et accessibles. Donc, $X \to v \in \mathcal{R}_2$ et $S \to_{G_2}^* u$. ■

Remarque: l'ordre des étapes est important car éliminer une règle contenant un non-terminal inaccessible X ne peut pas rendre improductif un non-terminal Y productif et accessible (si, dans une séquence $S \to^* aYb \to^* avb$, on utilise une règle $X \to u$, alors X est accessible). Par contre, éliminer une règle contenant

un non-terminal improductif peut rendre inaccessible un non-terminal productif et accessible. Par exemple, avec $\mathcal{R} = \{S \to aA \mid bBB \mid abAS \mid BC, A \to aB \mid bA \mid a, B \to aB \mid bB, C \to aA \mid bS \mid a\}$, nous avons $\mathcal{P} = \{A, C\} \cup \{S\}$ (B est improductif), $\mathcal{R}_1 = \{S \to aA \mid abAS, A \to bA \mid a, C \to aA \mid bS \mid a\}$, $\mathcal{A} = \{S\} \cup \{A\}$ (C est inaccessible) et donc $\mathcal{R}_2 = \{S \to aA \mid abAS, A \to bA \mid a\}$. Par contre, si nous avions fait l'étape 2 avant l'étape 1, nous aurions obtenu $\mathcal{A} = \mathcal{N}$ (tous les non-terminaux sont accessibles), $\mathcal{R}_1 = \mathcal{R}$, $\mathcal{P} = \{A, C\} \cup \{S\}$ (B est improductif) et $\mathcal{R}_2 = \{S \to aA \mid abAS, A \to bA \mid a, C \to aA \mid bS \mid a\}$ où C est inaccessible.

Lemme 7 Toute grammaire algébrique est effectivement équivalente à une grammaire propre et réduite.

Preuve. Soit G une grammaire algébrique. Par le lemme 5, G est effectivement équivalente à une grammaire algébrique propre G_1. Par le lemme 6, G_1 est effectivement équivalente à une grammaire algébrique réduite G_2 dont les règles forment un sous-ensemble des règles de G_1. Donc, G_2 est également propre. ■

Remarque: l'ordre est important. Il faut d'abord rendre la grammaire propre puis la réduire car, d'une part, réduire une grammaire ne fait qu'éliminer des règles et donc préserve le fait d'être propre, et d'autre part, rendre une grammaire propre peut rendre certains symboles inutiles. Par exemple, $\{S \to XY, X \to \varepsilon, Y \to \varepsilon\}$ est réduite et non propre. En la rendant propre, on obtient $\{S \to \varepsilon \mid XY\}$ où X et Y sont inutiles.

3.5 Propriétés des langages algébriques

Lemme 8 La classe Alg_Σ des langages algébriques sur un alphabet Σ est close par union, concaténation et étoile.

Preuve. Soit $G_1 = (\Sigma, \mathcal{N}_1, \mathcal{R}_1, S_1)$ et $G_1 = (\Sigma, \mathcal{N}_1, \mathcal{R}_1, S_1)$ deux grammaires algébriques. Sans perte de généralité (en renommant les non-terminaux si nécessaire), on peut supposer $\mathcal{N}_1 \cap \mathcal{N}_2 = \emptyset$. Soit S un nouveau non-terminal n'appartenant pas à $\mathcal{N}_1 \cup \mathcal{N}_2$. Alors, $L(G_1) \cup L(G_2) = L(G)$ où $G = (\Sigma, \mathcal{N}_1 \cup \mathcal{N}_2 \cup \{S\}, \mathcal{R}_1 \cup \mathcal{R}_2 \cup \{S \to S_1 \mid S_2\}, S)$ est algébrique; $L(G_1)L(G_2) = L(G)$ où $G = (\Sigma, \mathcal{N}_1 \cup \mathcal{N}_2 \cup \{S\}, \mathcal{R}_1 \cup \mathcal{R}_2 \cup \{S \to S_1 S_2\}, S)$ est algébrique; $L(G_1)^* = L(G)$ où $G = (\Sigma, \mathcal{N}_1 \cup \{S\}, \mathcal{R}_1 \cup \{S \to \varepsilon \mid S_1 S\}, S)$ est algébrique. ■

Lemme 9 (Pompage) Pour tout langage algébrique L, il existe un entier $N \geq 0$ tel que, pour tout mot $u \in L$ de longueur $|u| \geq N$, il existe des mots a, b, c, d, e tels que $u = abcde$, $bd \neq \varepsilon$, $|bcd| < N$ et, pour tout $i \geq 0$, $ab^i cd^i e \in L$.

Preuve. On procède par récurrence sur $|u|$.

Soit m la longueur maximale d'un membre droit de règle. C'est le nombre maximal de sous-arbres à chaque noeud d'un arbre de dérivation. Un arbre de hauteur i et de degré m a au plus m^i feuilles (récurrence sur i). Soit alors

$N = m^k + 1$ où k est le nombre de non-terminaux. Si $|u| \geq N$, alors u admet un arbre de dérivation de hauteur $> k$. Il existe donc une branche contenant deux fois le même non-terminal A. Ainsi, nous avons $S \to^* aAe$, $A \to^+ bAd$ et $A \to^* c$. Donc, pour tout $i \geq 0$, $ab^icd^ie \in L$.

On obtient $bd \neq \varepsilon$ en prenant une grammaire propre.

Si $|bcd| \geq N$ alors on peut conclure par hypothèse de récurrence (en prenant A comme non-terminal de départ) car $A \to^* bcd$ et $|bcd| < |u|$. ■

Remarque: si L est un ensemble fini de j mots (donc algébrique), alors le lemme est trivialement vérifié en prenant $N = j + 1$.

Ce lemme peut servir à montrer qu'un langage n'est pas algébrique. Par exemple, $L = \{a^nb^nc^n \mid n \in \mathbb{N}\}$. Comme $L = L_1 \cap L_2$ où $L_1 = \{a^nb^nc^k \mid n \in \mathbb{N}, k \in \mathbb{N}\}$ et $L_2 = \{a^kb^nc^n \mid n \in \mathbb{N}, k \in \mathbb{N}\}$, et $\overline{L} = \overline{\overline{L_1} \cap \overline{L_2}}$, on peut en déduire:

Lemme 10 La classe Alg_Σ des langages algébriques sur un alphabet Σ n'est close ni par intersection ni par complémentaire.

3.6 Analyse syntaxique des langages algébriques

Etant donné une grammaire G et un mot u, l'analyse syntaxique consiste à déterminer si u appartient au langage engendré par G ou non ($u \in L(G)$?), c'est-à-dire, s'il existe une dérivation de S à u ($S \to^* u$?).

On distingue deux types d'analyse:

- l'analyse descendante consiste à trouver u à partir de S (e.g. JavaCC),
- l'analyse ascendante consiste à trouver S à partir de u en prenant les règles à l'envers (e.g. yacc, CUP).

Ce problème est décidable pour les grammaires algébriques ainsi:

1. Soit $n = |u|$. Si $n = 0$, alors calculer l'ensemble des non-terminaux effaçables $\mathcal{E}$ et vérifier si $S \in^? \mathcal{E}$.
2. Sinon, calculer à partir de G une grammaire propre et réduite G' pour $L(G) - \{\varepsilon\}$. Il suffit alors de retourner $u \in^? \widehat{L}_n$ où $\widehat{L}_n$ est l'ensemble des mots dérivables en au plus n étapes en prenant $\widehat{L}_0 = \{S\}$ et $\widehat{L}_{k+1} = \widehat{L}_k \cup \{v \mid \exists u \in \widehat{L}_k, u \to v\}$. En effet, comme aucun membre droit de règle n'est un non-terminal ou ε, chaque étape de réécriture accroît la taille du mot engendré d'au moins une lettre.

La complexité dans le pire cas est en $\mathcal{O}(n^r)$ où r est le nombre de règles.

Il existe de meilleurs algorithmes, de complexité en $\mathcal{O}(n^3)$ dans le pire cas: l'algorithme de Cocke-Younger-Kasami ou l'algorithme d'Earley.

Cours 2 (1h45)

---

4 Automates finis

Un *automate (fini)* A est un quintuplet $(Q, \Sigma, \Delta, I, F)$ où Q est un ensemble fini d'états, Σ est un alphabet (fini), $\Delta \subseteq Q \times \Sigma \times Q$ une relation de transition, $I \subseteq Q$ un ensemble d'états initiaux, et $F \subseteq Q$ un ensemble d'états finaux.

L'ensemble des mots reconnus par A à partir de l'état q est $L_A(q) = \{u \in \Sigma^* \mid \exists f \in F, q \xrightarrow{u} f\}$ où $\xrightarrow{u}$ est défini par récurrence sur u de la manière suivante:

- $\xrightarrow{\varepsilon}$ est la relation d'égalité sur Q,
- $\xrightarrow{a}$ est la relation sur Q telle que $q \xrightarrow{a} q'$ si $(q, a, q') \in \Delta$,
- $\xrightarrow{au} = \xrightarrow{a}\xrightarrow{u}$.

Soit alors $L(A) = \bigcup_{i \in I} L_A(i)$ l'ensemble des mots reconnus par A (à partir d'un état initial).

Un langage L est *reconnaissable* (par automate fini) s'il existe un automate A tel que $L = L(A)$.

Deux automates A et B sont équivalents si $L(A) = L(B)$.

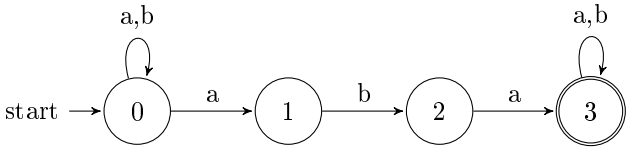
Soit $\Delta(q, a) = \{q' \in Q \mid q \xrightarrow{a} q'\}$ l'ensemble des états accessibles en lisant la lettre a à partir de l'état q.

Un automate peut-être représenté par la matrice $(\Delta(q, a))_{(q,a) \in Q \times \Sigma}$ dont les lignes sont indexées par Q et les colonnes par Σ (ou l'inverse) (voir Figure 1), ou par un graphe sur Q avec un arc entre p et q étiqueté par a ssi $p \xrightarrow{a} q$, un état initial étant souvent marqué par une flèche entrante et un état final par une flèche sortante ou un double cercle (voir Figure 2).

Figure 1: Automate 1 - Représentation tabulaire

$\Sigma \backslash Q$	0	1	2	3
a	0, 1		3	3
b	0	2		3

Figure 2: Automate 1 - Représentation graphique



4.1 Equivalence aux grammaires linéaires à droite

Une grammaire $G = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ est *linéaire à droite* si toute règle est de la forme $X \to \varepsilon$, $X \to a$ ou $X \to aY$, avec $a \in \Sigma$ et $X, Y \in \mathcal{N}$. Soit Lin_Σ la classe des grammaires linéaires à droite sur Σ. Il est clair qu'une grammaire linéaire à droite est algébrique. Nous allons voir que $Lin_\Sigma = Rec_\Sigma$. Ainsi, grammaires linéaires à droite, expression régulières et automates ont le même pouvoir expressif: ils définissent la même classe de langages.

Lemme 11 $Lin_\Sigma = Rec_\Sigma$

Preuve. Remarquons tout d'abord que toute grammaire linéaire à droite est équivalente à une grammaire linéaire à droite sans règle unaire: il suffit de remplacer toute règle $X \to a$ par les règles $X \to aX_a$ et $X_a \to \varepsilon$ où X_a est un nouveau non-terminal. Dans une telle grammaire $G = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, si $S \to^* u \in \Sigma^*$, alors tous les mots intermédiaires entre S et u sont de la forme vX avec $v \in \Sigma^*$ et $X \in \mathcal{N}$. On peut alors voir chaque non-terminal intermédiaire X comme un état, chaque règle de réécriture de la forme $X \to aY$ comme une transition $X \xrightarrow{a} Y$, et chaque non-terminal X ayant une règle $X \to \varepsilon$ comme un état terminal. On a ainsi $Lin_\Sigma \subseteq Rec_\Sigma$.

Montrons maintenant que $Rec_\Sigma \subseteq Lin_\Sigma$. Soit L un langage reconnaissable. Sans perte de généralité, nous pouvons supposer qu'il est reconnaissable par un automate déterministe $A = (Q, \Sigma, \delta, i, F)$. Soit alors la grammaire $G = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ telle que $\mathcal{N} = Q$, $S = i$ et $\mathcal{R} = \{p \to aq \mid \delta(p, a) = q\} \cup \{p \to \varepsilon \mid p \in F\}$. C'est bien une grammaire linéaire à droite reconnaissant le même langage. ■

4.2 Propriétés des langages reconnaissables par automate

Lemme 12 (Pompage) Pour tout langage reconnaissable L, il existe $N \geq 0$ tel que, pour tout $u \in L$, si $|u| \geq N$ alors il existe a, b, c tels que $u = abc$, $b \neq \varepsilon$, $|ab| \leq N$ et, pour tout $k \geq 0$, $ab^k c \in L$.

Preuve. Soit $A = (Q, \Sigma, \Delta, I, F)$ tel que $L = L(A)$. Prenons alors $N = |Q|$, le nombre d'états. Si $a_1 \dots a_n \in L$, alors il existe $q_1, \dots, q_{n+1} \in Q$ tels que, pour tout i, $q_i \xrightarrow{a_i} q_{i+1}$. Si $n \geq |Q|$ alors il existe $i < j$ tels que $q_i = q_j$ et $q_i, \dots, q_{j-1}$ sont distincts deux à deux. Ainsi il existe a, b, c tels que $b \neq \varepsilon$ et $q_0 \xrightarrow{a} q_i \xrightarrow{b} q_j \xrightarrow{c} q_{n+1}$. Donc, $|ab| \leq N$ et $ab^* c \subseteq L$. ■

Ce lemme nous permet de montrer que certains langages ne sont pas reconnaissables. Par exemple, $\{a^n b^n \mid n \in \mathbb{N}\}$. Ainsi, on ne peut pas utiliser un automate pour reconnaître un langage de programmation utilisant des parenthèses ou accolades, ou XML (a = parenthèse ouvrante, b = parenthèse fermante).

4.3 Transformations de base sur les automates

Un automate est *complet* si, pour tout q et pour tout a, $\Delta(q, a) \neq \emptyset$.

Lemme 13 Tout automate est effectivement équivalent à un automate complet.

Preuve. Il suffit de rajouter un nouvel état "puits"/"erreur" p, une transition $p \xrightarrow{a} p$ pour toute lettre a et, pour tout état q et lettre a tels que $\Delta(q,a) = \emptyset$, une transition $q \xrightarrow{a} p$.

Soit $A = (Q, \Sigma, \Delta, I, F)$ un automate. Soit $A' = (Q', \Sigma, \Delta', I, F)$ l'automate obtenu en prenant $Q' = Q \cup \{p\}$ où p est un élément quelconque n'appartenant pas à Q, $\Delta' = \Delta \cup \{(p,a,p) \mid a \in \Sigma\} \cup \{(q,a,p) \mid \Delta(q,a) = \emptyset\}$. A' est complet. Montrons maintenant que $L(A') = L(A)$. Si $u \in L(A)$, alors il existe $i \in I$ et $f \in F$ tels que $i \xrightarrow{u}_A f$. Comme $\xrightarrow{u}_A \subseteq \xrightarrow{u}_{A'}$, nous avons $u \in L(A')$. Réciproquement, soit $u = a_1 \dots a_n \in L(A')$. Par définition, il existe $q_0, \dots, q_n \in Q'$ tels que $q_0 \in I$, $q_n \in F$ et, pour tout $i \in \{1, \dots, n\}$, $q_i \in \Delta(q_{i-1}, a_i)$. Donc, pour tout i, $q_i \in Q$ et $u \in L(A)$. ■

Un état q est accessible s'il existe un état initial i et un mot u tel que $i \xrightarrow{u} q$. Un automate est *accessible* si tous ses états sont accessibles.

Lemme 14 Tout automate est effectivement équivalent à un automate accessible.

Preuve. Soit $A = (Q, \Sigma, \Delta, I, F)$ un automate. Soit $A' = (Q', \Sigma, \Delta', I', F')$ l'automate obtenu en prenant $Q' = \{q \in Q \mid \exists i \in I, \exists u \in \Sigma^*, i \xrightarrow{u}_A q\}$, $\Delta' = \Delta \cap Q' \times \Sigma \times Q'$, $I' = I \cap Q'$ et $F' = F \cap Q'$.

Soit $u = a_1 \dots a_n \in L(A)$. Par définition, il existe $q_0, \dots, q_n \in Q$ tels que $q_0 \in I$, $q_n \in F$ et, pour tout $i \in \{1, \dots, n\}$, $q_i \in \Delta(q_{i-1}, a_i)$. Donc, pour tout i, $q_i \in Q'$. Ainsi, $i \xrightarrow{u}_{A'} f$ et $u \in L(A')$.

Réciproquement, soit $u \in L(A')$. Alors, $u \in L(A)$ car $\xrightarrow{u}_{A'} \subseteq \xrightarrow{u}_A$. ■

Les deux lemmes précédents peuvent être combinés de telle sorte que:

Lemme 15 Tout automate est effectivement équivalent à un automate accessible et complet.

Un automate est *émondé* s'il est accessible et ne contient aucun état q inutile, c'est-à-dire, tel que $L(q) = \emptyset$. De même que toute grammaire algébrique peut-être réduite, tout automate peut-être émondé (prendre $Q' = \{q \in Q \mid \exists i \in I, \exists f \in F, \exists u, v \in \Sigma^*, i \xrightarrow{u}_A q \xrightarrow{v}_A f\}$ dans la preuve du lemme précédent).

Remarque: compléter un automate rajoute un état inutile, donc engendre un automate non émondé. Réciproquement, émonder un automate peut le rendre incomplet.

Un automate est *normalisé* s'il n'a qu'un seul état initial et un seul état final et si, de plus, l'état initial n'a aucune transition entrante et l'état final aucune transition sortante.

Remarque: l'automate réduit à un état à la fois initial et final, et n'ayant aucune transition, est l'unique automate normalisé reconnaissant $\{\varepsilon\}$.

Lemme 16 Pour tout automate A, il existe un automate normalisé A' tel que $L(A') = L(A) - \{\varepsilon\}$.

Preuve. Etant donné un automate $A = (Q, \Sigma, \Delta, I, F)$, soit $A' = (Q \cup \{i, f\}, \Sigma, \Delta', \{i\}, \{f\})$ où i et f sont deux nouveaux états et $\Delta' = \Delta \cup \{(i, a, q) \mid \exists p \in I, (p, a, q) \in \Delta\} \cup \{(p, a, f) \mid \exists q \in F, (p, a, q) \in \Delta\} \cup \{(i, a, f) \mid \exists p \in I, \exists q \in F, (p, a, q) \in \Delta\}$.

Si $u = a_1 \dots a_n \in L(A) - \{\varepsilon\}$, alors $n \geq 1$ et il existe $q_0, \dots, q_n$ tels que $q_0 \in I$, $q_n \in F$ et, pour tout $i \in \{1, \dots, n\}$, $q_{i-1} \xrightarrow{a_i}_A q_i$. Ainsi, nous avons $i \xrightarrow{a_1}_{A'} q_1 \overset{a_2 \dots a_{n-1}}{\longrightarrow}_{A'} q_{n-1} \xrightarrow{a_n} f$ et $u \in L(A')$.

Réciproquement, si $u = a_1 \dots a_n \in L(A')$, alors $n \geq 1$ (donc $u \neq \varepsilon$) et il existe $q_0, \dots, q_n$ tels que $q_0 = i$, $q_n = f$ et, pour tout $i \in \{1, \dots, n\}$, $q_{i-1} \xrightarrow{a_i}_{A'} q_i$. Ainsi, il existe $i' \in I$ et $f' \in F$ tels que $i' \xrightarrow{a_1} q_1$, $q_{n-1} \xrightarrow{a_n} f'$ et, pour tout $i \in \{2, \dots, n-1\}$, $q_i \notin \{i, f\}$ car i n'a pas de transition entrante et f n'a pas de transition sortante. Donc, $u \in L(A) - \{\varepsilon\}$. ■

4.4 Automates déterministes et déterminisation

Un automate est *déterministe* s'il n'a qu'un seul état initial et, pour tout q et pour tout a, $\Delta(q, a)$ a au plus un élément. Cela permet de définir l'état atteint (à partir de l'état initial) après avoir reconnu un mot u, par récurrence sur u:

- $\delta(q, \varepsilon) = q$,
- $\delta(q, a)$ est l'unique élément de $\Delta(q, a)$ (s'il existe),
- $\delta(q, au) = \delta(\delta(q, a), u)$.

Un automate déterministe peut être directement défini par un quintuplet $(Q, \Sigma, \delta, i, F)$ où $\delta : Q \times \Sigma \to Q$ et $i \in Q$.

Lemme 17 Tout automate est effectivement équivalent à un automate déterministe.

Preuve. Soit $A = (Q, \Sigma, \Delta, I, F)$ un automate non-déterministe. L'idée est de construire un automate déterministe (complet) $A' = (Q', \Sigma, \delta', i', F')$ dont les états sont des sous-ensembles de Q en prenant $Q' = \wp(Q)$, $\delta'(X, a) = \bigcup_{q \in X} \Delta(q, a)$, $i' = I$, et $F' = \{X \in Q' \mid X \cap F \neq \emptyset\}$.

Montrons que $L(A) \subseteq L(A')$. Soit $a_1 \dots a_n \in L(A)$. Alors il existe $q_0, \dots, q_n \in Q$ tels que $q_0 \in I$, $q_n \in F$ et, pour tout $k < n$, $q_{k+1} \in \Delta(q_k, a_{k+1})$. Montrons que, pour tout k, $q_k \in q'_k$, par récurrence sur k. Tout d'abord, $q'_0 = i' = I$ et $q_0 \in I$. Montrons alors que $q_{k+1} \in q'_{k+1}$ si $q_k \in q'_k$. Nous avons $q'_{k+1} = \delta'(q'_k, a_{k+1}) = \bigcup_{q \in q'_k} \Delta(q, a_{k+1})$. Donc $q_{k+1} \in q'_{k+1}$ car $q_{k+1} \in \Delta(q_k, a_{k+1})$ et, par hypothèse de récurrence, $q_k \in q'_k$.

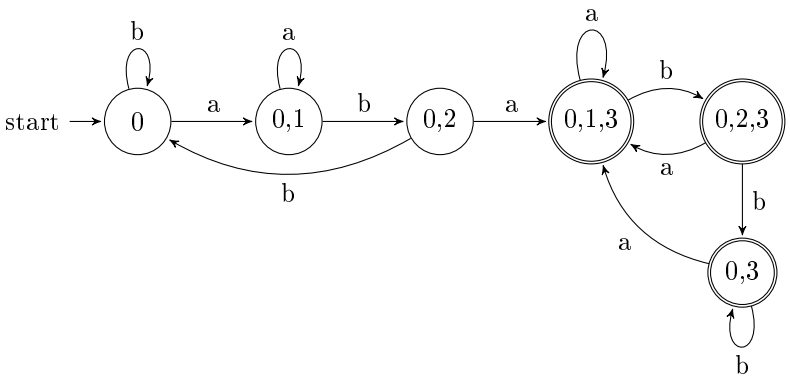
Montrons finalement que $L(A') \subseteq L(A)$. Soit $a_1 \dots a_n \in L(A')$. Alors il existe $q'_0, \dots, q'_n \in Q'$ tels que $q'_0 = i'$, $q'_n \in F'$ et, pour tout $k < n$, $q_{k+1} = \delta'(q_k, a_{k+1})$. Montrons qu'il existe $q_n, \dots, q_{n-k} \in Q$ tels que, pour

tout $j \leq k$, $q_{n-j} \in q'_{n-j}$, par récurrence sur k. Cas $k = 0$: comme $q'_n \in F'$, $q'_n \cap F \neq \emptyset$ et il existe $q_n \in q'_n \cap F$. Cas $k > 0$: supposons maintenant qu'il existe $q_n, \ldots, q_{n-k} \in Q$ tels que, pour tout $j \leq k$, $q_{n-j} \in q'_{n-j}$. Nous avons $q_{n-k} \in q'_{n-k} = \delta'(q'_{n-(k+1)}, a_{n-k}) = \bigcup_{q \in q'_{n-(k+1)}} \Delta(q, a_{n-k})$. Donc, il existe $q_{n-(k+1)} \in q'_{n-(k+1)}$. A la fin, nous avons $q_0 \in q'_0 = i' = I$. Donc, $a_1 \ldots a_n \in L(A)$. ■

Exemple: la déterminisation de l'automate de la Figure 1 donne l'automate de la Figure 3. On construit l'automate des parties colonne par colonne de gauche à droite:

$\Sigma \backslash \mathcal{P}(Q)$	0	$0,1$	$0,2$	$0,1,3$	$0,2,3$	$0,3$
a	$0,1$	$0,1$	$0,1,3$	$0,1,3$	$0,1,3$	$0,1,3$
b	0	$0,2$	0	$0,2,3$	$0,3$	$0,3$

Figure 3: Automate 2: déterminisé de l'automate 2



Cours 3 (1h45)

4.5 Automates avec ε-transitions et leur élimination

Un automate avec ε-transitions est un automate sur l'alphabet $\Sigma \cup \{\varepsilon\}$, c'est-à-dire, qu'il peut changer d'état sans générer de nouvelle lettre.

Lemme 18 Tout automate avec ε-transitions est équivalent à un automate sans ε-transitions.

Preuve. Soit $A = (Q, \Sigma, \Delta, I, F)$ un automate avec ε-transitions. On peut alors prendre $A' = (Q, \Sigma, \Delta', I, F)$ avec $\Delta' = \{(p, a, q) \in Q \times \Sigma \times Q \mid p(\xrightarrow{\varepsilon})^* \xrightarrow{a} (\xrightarrow{\varepsilon})^* q\}$.

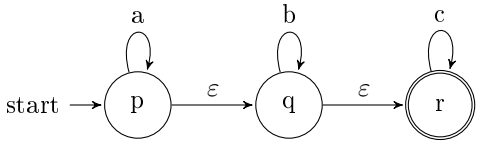
Mais on peut optimiser un peu en prenant la "fermeture avant" $(Q, \Sigma, \Delta', I', F)$ avec $\Delta' = \{(p, a, q) \mid p \xrightarrow{a} (\xrightarrow{\varepsilon})^* q\}$ et $I' = \{p \mid \exists i \in I, i \xrightarrow{\varepsilon}{}^* p\}$, ou la "fermeture arrière" $(Q, \Sigma, \Delta', I, F')$ avec $\Delta' = \{(p, a, q) \mid p(\xrightarrow{\varepsilon})^* \xrightarrow{a} q\}$ et $F' = \{p \mid \exists f \in F, p \xrightarrow{\varepsilon}{}^* f\}$.

Montrons la correction de la fermeture avant.

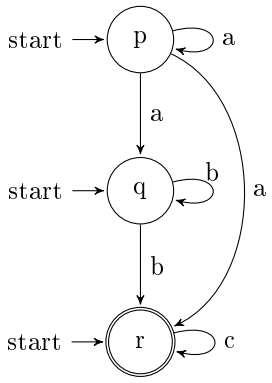
Montrons que, si $u = a_1 \dots a_n \in L(A)$, alors $u \in L(A')$. Par définition, si $u \in L(A)$, alors il existe des états $q_0, q'_0, \dots, q_n, q'_n$ tels que $q_0 \in I$, $q_i(\xrightarrow{\varepsilon}_A)^* q'_i$, $q'_i \xrightarrow{a_{i+1}}_A q_{i+1}$ et $q'_n \in F$. Ainsi, $q'_0 \in I'$, $q'_i \xrightarrow{a_1}_{A'} q'_{i+1}$ et $q'_n \in F$. Donc, $u \in L(A')$.

Réciproquement, si $u \in L(A')$, alors il existe $q'_0, \dots, q'_n$ tels que $q'_0 \in I'$, $q'_i \xrightarrow{a_{i+1}}_{A'} q'_{i+1}$ et $q'_n \in F$. Par définition de I' et $\longrightarrow_{A'}$, il existe q_i ($q_0 \in I$) tels que $q_i(\xrightarrow{\varepsilon}_A)^* q'_i$ et $q'_i \xrightarrow{a_{i+1}}_A q_{i+1}$. Donc $u \in L(A)$. ■

Exemple. Eliminons les ε-transitions de l'automate suivant en calculant sa fermeture avant:



1. On calcule les ε-réduits: $p \xrightarrow{\varepsilon} q \xrightarrow{\varepsilon} r$.
2. On rajoute dans les états initiaux tous les réduits d'un état initial. Ici, on obtient $\{p, q, r\}$ initiaux.
3. Pour chaque transition $\alpha \xrightarrow{x} \beta$ et état β' tel que $\beta \xrightarrow{\varepsilon}{}^* \beta'$, on rajoute une transition $\alpha \xrightarrow{x} \beta'$. Ici:
   - pour $p \xrightarrow{a} p$, on rajoute $p \xrightarrow{a} q$ et $p \xrightarrow{a} r$;
   - pour $q \xrightarrow{b} q$, on rajoute $q \xrightarrow{a} r$.
4. On élimine toutes les ε-transitions.
5. On obtient ainsi:



4.6 Propriétés de clôture des langages reconnaissables

En utilisant des ε-transitions, il est facile de voir que:

Lemme 19 La classe Rec_Σ des langages sur un alphabet Σ reconnaissables par automate fini est close par union, concaténation, étoile, complémentaire et intersection.

Preuve. Soient deux automates $A_1 = (Q_1, \Sigma, \Delta_1, I_1, F_1)$ et $A_2 = (Q_2, \Sigma, \Delta_2, I_2, F_2)$. Sans perte de généralité, on peut supposer que $Q_1 \cap Q_2 = \emptyset$. Dans le cas contraire, il suffit de "renommer" les états de l'un des automates.

Pour reconnaître $L(A_1) \cup L(A_2)$, il suffit de prendre $A_1 \cup A_2 = (Q_1 \cup Q_2, \Sigma, \Delta_1 \cup \Delta_2, I_1 \cup I_2, F_1 \cup F_2)$.

Pour reconnaître $L(A_1)L(A_2)$, il suffit de prendre l'automate avec ε-transitions $A_1A_2 = (Q_1 \cup Q_2, \Sigma, \Delta_1 \cup \Delta_2 \cup \{f_1 \xrightarrow{\varepsilon} i_2 \mid f_1 \in F_1, i_2 \in I_2\}, I_1, F_2)$.

Pour reconnaître $L(A_1)^*$, il suffit de prendre l'automate avec ε-transitions $A_1^* = (Q_1, \Sigma, \Delta_1 \cup \{f \xrightarrow{\varepsilon} i \mid f \in F_1, i \in I_1\}, I_1, F_1)$.

Pour reconnaître $\overline{L(A_1)} = \Sigma^* - L(A_1)$, le complémentaire de A_1, nous pouvons supposer sans perte de généralité que A_1 est déterministe et complet. Il suffit alors de prendre $\overline{A_1} = (Q, \Sigma, \Delta, I, Q - F)$. En effet, si $u \in L(\overline{A_1})$, alors il existe $i \in I$ et $q \in Q - F$ tels que $i \xrightarrow{u} q$. Comme A_1 est déterministe, i et q sont uniques. Donc, $u \in \overline{L(A_1)}$. Réciproquement, si $u \in \overline{L(A_1)}$ alors, comme A_1 est complet, il existe $q \in Q - F$ tel que $i \xrightarrow{u} q$. Donc, $u \in L(\overline{A_1})$.

Prendre un automate déterministe et complet est important comme le montrent les contre-exemples suivants:

- L'automate incomplet $\longrightarrow 0 \xrightarrow{a} 1$ avec 1 final reconnaît $\{a\}$. L'automate complémentaire $\longrightarrow 0 \xrightarrow{a} 1$ avec 0 final reconnaît $\{\varepsilon\}$ et non pas $\Sigma^* - \{a\}$.

- L'automate non-déterministe $\longrightarrow 0 \xrightarrow{a} 1, 2$ avec 1 final reconnaît $\{a\}$. L'automate complémentaire $\longrightarrow 0 \xrightarrow{a} 1, 2$ avec $0, 2$ finaux reconnaît $\{\varepsilon, a\}$ et non pas $\Sigma^* - \{a\}$.

Enfin, pour reconnaître $L(A_1) \cap L(A_2)$, il suffit de prendre l'automate reconnaissant $\overline{\overline{L(A_1)} \cup \overline{L(A_2)}}$, c'est-à-dire $\overline{A}$ où A est le déterminisé et le complété de $\overline{A_1} \cup \overline{A_2}$, à supposer que A_1 et A_2 soient eux-mêmes déjà déterminisés et complétés. Une autre manière, plus simple, est de prendre l'automate $A_1 \times A_2 = (Q_1 \times Q_2, \Sigma, \Delta, I_1 \times I_2, F_1 \times F_2)$ où $\Delta = \{((p_1, p_2), a, (q_1, q_2)) \mid (p_1, a, q_1) \in \Delta_1, (p_2, a, q_2) \in \Delta_2\}$. ■

4.7 Minimisation du nombre d'états

Petits rappels mathématiques préliminaires:

Etant donné un ensemble A, l'ensemble des parties ou sous-ensembles de A, est noté $\mathcal{P}(A)$.

Une *relation* sur un ensemble A est un ensemble R de paires (x, y) avec $x, y \in A$, c'est-à-dire, une partie de $A \times A$. On note souvent $(x, y) \in R$ par xRy.

Si, pour tout $x \in A$, xRx, R est dite *réflexive*. Si, pour tout $x, y \in A$, xRy implique yRx, R est dite *symétrique*. Si, pour tout $x, y, z \in A$, xRy et yRz impliquent xRz, R est dite *transitive*. Enfin, si R est à la fois réflexive, symétrique et transitive, alors R est dite une *relation d'équivalence*. Par exemple, l'égalité est une relation d'équivalence.

Etant donné une relation d'équivalence $\simeq$ sur un ensemble A, la *classe d'équivalence* d'un élément $x \in A$ est l'ensemble (non vide) $[x] = \{y \in A \mid y \simeq x\}$ des éléments équivalents à x. On note par $A/\simeq$ l'ensemble des classes d'équivalences. Les classes d'équivalence étant disjointes, elles forment une partition de A: nous avons $\bigcup A/\simeq = A$.

Nous avons vu plusieurs transformations augmentant le nombre d'états. Inversement, est-il possible de réduire le nombre d'états d'un automate tout en préservant le langage reconnu? Nous allons voir que l'ensemble des automates déterministes complets d'un langage reconnaissable admet un automate minimal par rapport au nombre d'états.

Le *résiduel* (à gauche) d'un langage L par rapport à un mot u est le langage $u^{-1}L = \{v \in \Sigma^* \mid uv \in L\}$, c'est-à-dire l'ensemble des mots v tel que $uv \in L$.

Lemme 20 Pour tout langage reconnaissable L et tout automate déterministe accessible complet $A = (Q, \Sigma, \delta, i, F)$ reconnaissant L, $\{L_A(q) \mid q \in Q\} = \{u^{-1}L \mid u \in \Sigma^*\}$.

Preuve. Montrons d'abord que, si $i \xrightarrow{u} q$, alors $u^{-1}L = L_A(q)$. Soit $v \in u^{-1}L$. Alors $uv \in L$. Comme A est déterministe, nous avons $i \xrightarrow{u} q \xrightarrow{v} f \in F$. Donc, $v \in L_A(q)$. Réciproquement, si $v \in L_A(q)$, alors $q \xrightarrow{v} f \in F$ et $uv \in L$.

Réciproquement, soit $u \in \Sigma^*$. Comme A est complet, il existe q tel que $i \xrightarrow{u} q$. Donc, $u^{-1}L = L_A(q)$. ■

Ainsi, l'ensemble $\{L_A(q) \mid q \in Q\}$ est indépendant de A et ne dépend que du langage reconnu par A: c'est l'ensemble de ses résiduels. De plus, comme Q est fini, l'ensemble des résiduels est fini, et le nombre de résiduels est une borne

inférieure au nombre d'états de tout automate déterministe accessible complet reconnaissant L. En fait, cela caractérise les langages reconnaissables:

Lemme 21 Un langage est reconnaissable si et seulement si l'ensemble de ses résiduels est fini.

Preuve. Soit L un langage. Nous avons déjà vu que, si L est reconnaissable, alors l'ensemble de ses résiduels est fini (car tout automate est équivalent à un automate déterministe complet dont tous les états sont accessibles). Montrons maintenant que, si l'ensemble des résiduels est fini, alors L est reconnaissable. L'idée est d'utiliser l'ensemble des résiduels comme ensemble d'états en prenant $A = (Q, \Sigma, \delta, i, F)$ où $Q = \{u^{-1}L \mid u \in \Sigma^*\}$, $\delta(q, a) = a^{-1}q$, $i = \varepsilon^{-1}L = L$ et $F = \{u^{-1}L \mid u \in L\}$.

Montrons d'abord que, pour tout u, $\delta(L, u) = u^{-1}L$ (l'état atteint après avoir reconnu u est $u^{-1}L$), par récurrence sur la longueur de u. Si $u = \varepsilon$, $\delta(L, \varepsilon) = L = \varepsilon^{-1}L$. Si $u = a \in \Sigma$, $\delta(L, a) = a^{-1}L$. Enfin, si $u = av$, $\delta(L, av) = \delta(\delta(L, a), v) = \delta(a^{-1}L, v) = v^{-1}(a^{-1}L)$ par hypothèse de récurrence, et $v^{-1}(a^{-1}L) = (av)^{-1}L$.

Montrons maintenant que $L(A) \subseteq L$. Soit $u \in L(A)$. Alors, $\delta(i, u) = u^{-1}L \in F$ et donc $u \in L$ par définition de F.

Montrons maintenant que $L \subseteq L(A)$. Soit $u \in L$. Alors, $\delta(i, u) = u^{-1}L$. Comme $u \in L$, nous avons $u^{-1}L \in F$ et donc $u \in L(A)$. ■

Corollaire 2 L'automate construit dans la preuve précédente est déterministe, accessible, complet et minimal par rapport au nombre d'états parmi tous les automates déterministes, accessibles et complets reconnaissant le même langage.

Preuve. Soit L un langage reconnaissable. Alors, l'ensemble R des résiduels est fini. L'automate A construit dans la preuve précédente est déterministe accessible complet et a R comme ensemble d'états. Soit maintenant B un automate déterministe accessible complet reconnaissant L et ayant un ensemble d'états Q. Alors, $|R| = |\{L_A(q) \mid q \in Q, i \to^* q\}| \leq |Q|$. ■

Nous avons montré qu'il existe un automate de taille minimal. Comment le calculer effectivement? Nous avons vu que, pour un automate donné, l'ensemble des langages générés à partir d'un état accessible est égal à l'ensemble des résiduels. Cela suggère de fusionner tous les états générant le même langage. Problème: comment savoir si deux états génèrent le même langage?

On suppose donné un automate $(Q, \Sigma, \delta, i, F)$. On dit que deux états p et q sont équivalents, $p \simeq q$, si $L(p) = L(q)$. Un automate minimal est également donné par $(Q', \Sigma, \delta', i', F')$ où $Q' = Q/\simeq$ est l'ensemble des classes d'équivalences modulo $\simeq$, $\delta'([q], a) = [\delta(q, a)]$, $[i]$ est la classe d'équivalence de i, et $F' = \{X \in Q' \mid X \cap F \neq \emptyset\}$ est l'ensemble des classes d'équivalence contenant un état final. La fonction δ' est bien définie car $\delta(q, a) \simeq \delta(q', a)$ dès que $q \simeq q'$ $(L(\delta(q, a)) = a^{-1}L(q))$. Problème: comment calculer $\simeq$?

On remarque que $\simeq = F(\simeq)$ où $F(\mathcal{Y}) = \{(p, q) \in \mathcal{Y} \mid \forall a \in \Sigma, (\delta(p, a), \delta(q, a)) \in \mathcal{Y}\}$. Soit E l'ensemble des classes d'équivalence sur Q équipé de l'ordre $\mathcal{Y} \leq \mathcal{Z}$

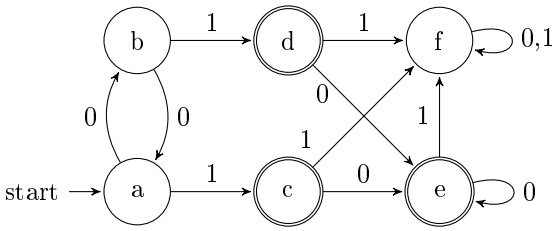
ssi $\mathcal{Z} \subseteq \mathcal{Y}$. E est fini car Q est fini. On peut vérifier que F est monotone sur E. Soit $A_0 = \{(p,q) \in Q^2 \mid \{p,q\} \subseteq F \vee \{p,q\} \subseteq Q - F\}$ la relation d'équivalence telle que p et q sont équivalents s'ils sont tous les deux finaux ou non-finaux. Soit alors E' l'ensemble des classes d'équivalence plus grandes que A_0, c'est-à-dire incluses dans A_0. Nous avons $\simeq \subseteq A_0$ donc $\simeq \in E'$. Ainsi, d'après le théorème de Tarski, F admet une plus petite solution $A_0 \leq A \leq \simeq$, c'est-à-dire $\simeq \subseteq A \subseteq A_0$.

Etant donné $k \in \mathbb{N}$, on dit que deux états p et q sont k-équivalents, $p \simeq_k q$, si $L_k(p) = L_k(q)$ où $L_k(p) = \{u \in L(p) \mid |u| \leq k\}$ est l'ensemble des mots reconnus à partir de q de longueur au plus k. Ainsi, $p \simeq q$ ssi, pour tout k, $p \simeq_k q$, c'est-à-dire:

$$\simeq = \bigcap_{k \in \mathbb{N}} \simeq_k.$$

On peut alors montrer par récurrence sur k que $\simeq_k \subseteq A$. Il s'ensuit que $\simeq = A$ et donc que $\simeq$ peut être calculé par itération de F à partir de A_0.

Figure 4: Automate 3



Par exemple, minimisons l'automate déterministe accessible complet de la Figure 4 (pris sur Wikipedia). Commençons par écrire sa table de transition:

	a	b	c	d	e	f
0	b	a	e	e	e	f
1	c	d	f	f	f	f

Au début, nous avons $Q/\simeq_0 = \{A, B\}$ où $A = \{c, d, e\}$ est l'ensemble des états finaux et $B = \{a, b, f\}$ est l'ensemble des états non finaux. Pour voir le comportement de chaque état, on remplace dans la table de transition chaque état par sa classe d'équivalence modulo $\simeq_0$:

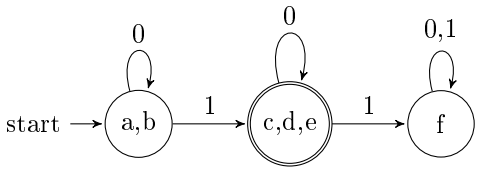
	a	b	c	d	e	f
0	B	B	A	A	A	B
1	A	A	B	B	B	B

On voit qu'il y a trois comportements différents. Ainsi, $Q/\simeq_1 = \{A, C, D\}$ où $C = \{f\}$ et $D = \{a, b\}$. C ne peut pas être raffiné davantage car c'est un singleton. Maintenant, on remplace dans la table de transition chaque état par sa classe d'équivalence modulo $\simeq_1$:

	a	b	c	d	e
0	D	D	A	A	A
1	A	A	C	C	C

On a donc atteint le point fixe: $Q/\simeq_2 = Q/\simeq_1$. On voit donc que c, d et e peuvent être fusionnés en un seul état, ainsi que a et b. On obtient l'automate déterministe accessible complet minimal de la Figure 5.

Figure 5: Automate 4: minimisation de l'automate 4



Cours 4 (1h45)

---

5 Expressions régulières

L'ensemble des *expressions régulières* sur un alphabet Σ est le plus petit ensemble $\mathcal{E}$ tel que:

- $\{\} \in \mathcal{E}$;
- $\Sigma \subseteq \mathcal{E}$;
- si $e_1, e_2 \in \mathcal{E}$, alors $e_1 \,|\, e_2 \in \mathcal{E}$ et $e_1 \,;\, e_2 \in \mathcal{E}$;
- si $e \in \mathcal{E}$, alors $e* \in \mathcal{E}$.

C'est le plus petit ensemble d'arbres dont les noeuds sont étiquetés par un élément de $\Sigma \cup \{\{\}, |, ;, *\}$ et tels que, si un arbre est étiqueté par un élément de $\Sigma \cup \{\{\}\}$ alors il n'a pas de sous-arbre, si un arbre est étiqueté par $|$ ou $;$, alors il a deux sous-arbres, et si un arbre est étiqueté par $*$, alors il a un sous-arbre.

Toute expression régulière e peut-être interprétée par un langage $L(e)$ définissable par récurrence sur e:

- $L(\{\}) = \emptyset$
- $L(a) = \{a\}$
- $L(e_1 \,|\, e_2) = L(e_1) \cup L(e_2)$
- $L(e_1 \,;\, e_2) = L(e_1)L(e_2)$

- $L(e*) = L(e)^*$

Remarque: $L(\{\}*) = \{\varepsilon\}$.

Soit $Reg_\Sigma = \{L(e) \subseteq \Sigma^* \mid e \in \mathcal{E}\}$ l'ensemble des langages exprimables par une expression régulière.

5.1 Equivalence avec les automates

Nous avons vu que la classe Rec_Σ des langages reconnaissables par un automate est close par union, concaténation et étoile. Elle contient aussi $\emptyset$ et $\{a\}$ pour chaque lettre a. Donc, $Reg_\Sigma \subseteq Rec_\Sigma$.

Inversement, on peut se demander si tout langage reconnaissable par un automate ne peut pas être exprimé par une expression régulière. Pour cela, remarquons que, si $A = (Q, \Sigma, \Delta, I, F)$ est un automate et si on note $\cup$ par $+$, alors $L(A) = \sum_{i \in I} L(i)$ où:

$$L(p) = \sum_{p \xrightarrow{a} q} aL(q) + \{\varepsilon \mid p \in F\}$$

Remarque: $\{\varepsilon \mid p \in F\} = \{\varepsilon\}$ si $p \in F$, et $\emptyset$ sinon.

Autrement dit, $(L(p))_{p \in Q}$ est solution, en prenant $X_p = L(p)$, du système d'équations suivant:

$$X_p = \sum_{p \xrightarrow{a} q} aX_q + \{\varepsilon \mid p \in F\}$$

Lemme 22 (Arden) Soient K et L deux langages, et $\mathcal{S}$ l'ensemble des langages X tels que $X = KX + L$.

- Si $\varepsilon \in K$, alors il y a une infinité de solutions: $\mathcal{S} = \{K^*(L + Y) \mid Y \subseteq \Sigma^*\}$, et la plus petite pour l'inclusion est K^*L.
- Si $\varepsilon \notin K$, alors il n'y a qu'une seule solution: $\mathcal{S} = \{K^*L\}$.

Preuve. Remarquons d'abord que K^*L est toujours solution ($K^*L \in \mathcal{S}$). En effet, si $X = K^*L$, alors $KX + L = K^+L + L = K^*L = X$.

Montrons ensuite que, si $\varepsilon \in K$ alors, pour tout Y, $K^*(L + Y)$ est solution, c'est-à-dire, $\{K^*(L+Y) \mid Y \subseteq \Sigma^*\} \subseteq \mathcal{S}$. Si $X = K^*(L+Y) = K^*L+K^*Y$, alors $KX + L = K^+L + K^+Y + L = K^*L + K^+Y$. Reste à montrer que $K^+ = K^*$. D'une part, nous avons $K^+ \subseteq K^*$ puisque $K^* = \varepsilon + K^+$. D'autre part, comme $\varepsilon \in K$ et $K \subseteq K^+$, nous avons $K^* = \varepsilon + K^+ \subseteq K^+$. Donc, $K^+ = K^*$ et $KX + L = X$.

Montrons maintenant que tout $X \in \mathcal{S}$ est de la forme $K^*(L + Y)$.

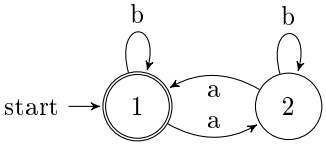
Montrons d'abord que, si X est solution de $X = KX + L$ alors, pour tout n, $X = K^nX + (\sum_{i=0}^{n-1} K^i)L$, par récurrence sur n. Pour $n = 0$, nous avons $K^0X + (\sum_{i=0}^{-1} K^i)L = X + \emptyset L = X$. Supposons maintenant que $X = K^nX + (\sum_{i=0}^{n-1} K^i)L$. Alors, $X = K^n(KX+L)+(\sum_{i=0}^{n-1} K^i)L = K^{n+1}X+(\sum_{i=0}^{n} K^i)L$.

Ainsi, pour tout n, $K^nX \subseteq X$ et $(\sum_{i=0}^{n-1} K^i)L \subseteq X$. Donc, $K^*L \subseteq X$. Soit alors $Y = X - K^*L$. Comme $Y \subseteq X$, nous avons $K^nY \subseteq K^nX \subseteq X$. Donc, $K^*Y \subseteq X$. Ainsi, $K^*(L+Y) \subseteq X = K^*L + Y \subseteq K^*(L+Y)$. Donc, $X = K^*(L+Y)$.

Il s'ensuit que, si $\varepsilon \in K$, alors $\mathcal{S} = \{K^*(L+Y) \mid Y \subseteq \Sigma^*\}$.

Il ne reste plus maintenant qu'à montrer que, si $\varepsilon \notin K$, alors $\mathcal{S} = \{K^*L\}$, c'est-à-dire, $Y = \emptyset$. Procédons par l'absurde. Supposons que $Y = X - K^*L \neq \emptyset$. Soit alors u un mot de Y de longueur minimale. Comme $u \notin K^*L$, nous avons $u \notin L$. Comme $u \in X - L$ et $X = KX + L$, nous avons $u = kv$ avec $k \in K$ et $v \in X$. Comme $u \notin K^*L$, nous avons également $v \notin K^*L$. Comme $\varepsilon \notin K$, $k \neq \varepsilon$ et v est plus petit que u. Contradiction. ■

Figure 6: Automate 5



Par exemple, considérons l'automate de la Figure 6. Nous avons:

$$\begin{cases} X_1 &= bX_1 + aX_2 + \varepsilon \\ X_2 &= aX_1 + bX_2 \end{cases}$$

En appliquant le lemme d'Arden sur la 2e équation, on obtient $X_2 = b^*aX_1$. En remplaçant X_2 par b^*aX_1 dans la 1ère équation (méthode de Gauss), on obtient $X_1 = bX_1 + ab^*aX_1 + \varepsilon$. Donc, par le lemme d'Arden à nouveau, on obtient $X_1 = (b + ab^*a)^*$.

On voit ainsi que, de proche en proche, on peut exprimer $L(A)$ par une expression régulière:

Théorème 2 (Kleene) $Rec_\Sigma = Reg_\Sigma$

Il existe d'autres manières d'exprimer le langage reconnu par un automate sous la forme d'une expression régulière.

Méthode de McNaughton-Yamada. Dans cette méthode, on suppose que $Q = \{1, \ldots, n\}$ où n est le nombre d'états. On remarque ensuite que $L = \bigcup_{i \in I, f \in F} L^n_{i,f}$ où $L^k_{p,q} = \{a_1 \ldots a_n \in \Sigma^* \mid n \geq 0, \exists k_1, \ldots, k_{n-1} \leq k, p \xrightarrow{a_1} k_1 \xrightarrow{a_2} k_2 \ \ldots k_{n-1} \xrightarrow{a_n} q\}$ est l'ensemble des mots reconnaissables en partant de p et en arrivant à q via des états intermédiaires $\leq k$. La méthode de McNaughton-Yamada consiste alors à calculer $L^n_{p,q}$ par récurrence de la façon suivante:

$$\begin{aligned} L^0_{p,q} &= \{a \in \Sigma \mid (p,a,q) \in \Delta\} \cup \{\varepsilon \mid p = q\} \\ L^{k+1}_{p,q} &= L^k_{p,q} + L^k_{p,k+1}(L^k_{k+1,k+1})^* L^k_{k+1,q} \end{aligned}$$

De même, on voit que L s'exprime par une expression régulière.

Méthode de Brzozowski-McCluskey. Dans cette méthode, on construit, à partir d'un automate $A = (Q, \Sigma, \Delta, I, F)$ dont les états sont numérotés de 1 à n, une séquence d'automates $A_n, \ldots, A_0$, où $A_k = (Q_k, \mathcal{E}, \Delta_k, \{i\}, \{f\})$, $Q_k = \{i, 1, \ldots, k, f\}$ et:

- $\Delta_n = \{(p, \Sigma_{p \xrightarrow{a} q} a, q) \mid p, q \in Q\} \cup \{(i, \varepsilon, p) \mid p \in I\} \cup \{(p, \varepsilon, f) \mid p \in F\}$,
- $\Delta_{k-1} = \{(p, a + bc^*d, q) \mid p, q \in Q_{k-1}, (p, a, q), (p, b, k), (k, c, k), (k, d, q) \in \Delta_k\}$.

Pour calculer Δ_{k-1}, on élimine dans Δ_k l'état k et, pour chaque paire d'états (p, q) différents de k, on remplace l'étiquette a de la transition entre p et q par $a + bc^*d$ où b est l'étiquette de la transition entre p et k, c est l'étiquette de la transition entre k et k, et d est l'étiquette de la transition entre k et q.

A la fin, on obtient un automate avec seulement deux états, i et f, et une seule transition entre i et f étiquetée par une expression régulière exprimant le langage de l'automate.

Cours 5 (1h45)

---

6 Grammaires $LL(k)$

Dans ce chapitre, on considère une classe de langages analysables par une analyse descendante consistant à regarder un nombre fixe k de lettres pour décider quelle règle appliquer.

Dans LL(k), le premier L signifie "left-to-right scanning" (lecture du mot de gauche à droite), le second L signifie "left-most derivation" (en réécrivant à chaque étape le non-terminal le plus à gauche), et k est le nombre de caractères suivants utilisés pour choisir quelle règle appliquée.

Le *préfixe de longueur k au plus*, ou k-préfixe, d'un mot u est défini par récurrence ainsi:

- $u|_0 = \varepsilon$,
- $\varepsilon|_{k+1} = \varepsilon$,
- $(au)|_{k+1} = a(u|_k)$.

Soit alors

$$Pre_k(u) = \{v|_k \mid u \to^* v \in \Sigma^*\}$$

l'ensemble des k-préfixes d'un mot de Σ^* dérivable à partir de u.

Remarque: $Pre_k(u) = \{u|_k\}$ si $u \in \Sigma^*$, et $Pre_k(u') \subseteq Pre_k(u)$ si $u \to u'$.

Soit

$$Pre_k(L) = \bigcup_{u \in L} Pre_k(u)$$

l'ensemble des k-préfixes d'un mot dérivable à partir d'un mot de L.

Remarque: Pre_k est monotone (si $L \subseteq L'$ alors $Pre_k(L) \subseteq Pre_k(L')$).

Définition 1 (Grammaire LL(k)) Une grammaire G est $LL(k)$ si, pour toutes dérivations $S \to_g^* uXv \to_g ur_1v \to_g^* ur'_1v'_1 \in \Sigma^*$ et $S \to_g^* uXv \to_g ur_2v \to_g^* ur'_2v'_2 \in \Sigma^*$ telles que $(r'_1v'_1)|_k = (r'_2v'_2)|_k$, alors $r_1 = r_2$. Un langage L est $LL(k)$ s'il existe une grammaire G $LL(k)$ tel que $L = L(G)$.

Soit $LL_\Sigma(k)$ l'ensemble des langages sur Σ ayant une grammaire $LL(k)$.

Un langage est $LL(0)$ si et seulement si c'est le langage vide $\emptyset$ ou un langage réduit à un mot (exercice).

Tout langage régulier est $LL(1)$ (exercice). Mais pas tout langage $LL(1)$ est régulier. Par exemple, le langage $\{a^nb^n \mid n \in \mathbb{N}\}$ est $LL(1)$ mais non régulier. Ainsi, $Reg_\Sigma \subsetneq LL_\Sigma(1)$.

Nous avons $LL_\Sigma(k) \subsetneq LL_\Sigma(k+1)$. En effet, $\{a^kb, a^kc\}$ est $LL(k+1)$ mais pas $LL(k)$.

Cependant, pas tous les langages algébriques sont $LL(k)$ pour un certain k: $\bigcup_{k\in\mathbb{N}} LL_\Sigma(k) \subsetneq Alg_\Sigma$. Par exemple, $\{a^n0b^n \mid n \in \mathbb{N}\} \cup \{a^n1b^{2n} \mid n \in \mathbb{N}\}$ est algébrique mais n'appartient à aucun $LL_\Sigma(k)$.

Savoir s'il existe k tel qu'une grammaire soit $LL(k)$ est décidable. Par contre, savoir s'il existe une grammaire $LL(k)$ pour un langage donné n'est pas décidable.

6.1 Propriétés des grammaires $LL(k)$

Nous allons maintenant voir quelques propriétés des grammaires $LL(k)$.

Soit

$$\widehat{Suiv}(X) = \{v \mid \exists u, S \to_g^* uXv\}$$

l'ensemble des mots apparaissant après X dans une dérivation, et

$$Suiv_k(X) = Pre_k(\widehat{Suiv}(X))$$

l'ensemble des k-préfixes des mots de Σ^* dérivables à partir d'un mot suivant X.

Lemme 23 (Propriétés des grammaires $LL(k)$)

1. Une grammaire est $LL(k)$ ssi, pour toutes règles $X \to r_1 \neq X \to r_2$ et mots $v \in \widehat{Suiv}(X)$, nous avons $Pre_k(r_1v) \cap Pre_k(r_2v) = \emptyset$.
2. Dans une grammaire $LL(k)$ avec $k \geq 1$, il n'y a pas deux règles $X \to r_1 \neq X \to r_2$ tels que r_1 et r_2 aient un préfixe commun non vide.
3. Une grammaire $LL(k)$ est non-ambiguë.
4. Une grammaire $LL(k)$ réduite n'est pas directement récursive à gauche, c'est-à-dire, ne contient aucune règle de la forme $X \to Xs$.

Preuve.

1. Par contraposée. Si G n'est pas $LL(k)$, alors il existe $X \to r_1 \neq X \to r_2$ telles que $S \to_g^* uXv \to_g ur_1v \to_g^* ur_1'v_1' \in \Sigma^*$, $S \to_g^* uXv \to_g ur_2v \to_g^* ur_2'v_2' \in \Sigma^*$ et $(r_1'v_1')|_k = (r_2'v_2')|_k$. Or $(r_i'v_i')|_k \in Pre_k(r_iv)$. Donc, $Pre_k(r_1v) \cap Pre_k(r_2v) \neq \emptyset$. Réciproquement, supposons qu'il existe $X \to r_1 \neq X \to r_2$ et $v \in \widehat{Suiv}(X)$ tels que $Pre_k(r_1v) \cap Pre_k(r_2v) \neq \emptyset$. Alors, il existe $r_1', v_1', r_2', v_2' \in \Sigma^*$ tels que $r_i \to^* r_i'$, $v \to^* v_i'$ et $(r_1'v_1')|_k = (r_2'v_2')|_k$. Donc G n'est pas $LL(k)$.

2. Immédiat.

3. Par contradiction. Soit G une grammaire $LL(k)$ ambiguë. Alors, il existe un mot w ayant deux arbres de dérivations différents, donc deux dérivations gauche différentes $S \to_g^* uXv \to_g ur_1v \to_g^* ur_1'v' = w$ et $S \to_g^* uXv \to_g ur_2v \to_g^* ur_2'v' = w$ telles que $r_1 \neq r_2$. Or, $r_1'v' = r_2'v'$. Contradiction.

4. Par contradiction. Soit G une grammaire $LL(k)$ réduite avec une règle de la forme $X \to Xs$. Comme la grammaire est réduite, il doit aussi y avoir une règle de la forme $X \to r$ avec aucun X au début de r. Autrement, X serait improductif. De plus, il existe u et v tels que $S \to_g^* uXv$. En appliquant la première règle n fois, on obtient $uXv \to_g^n uXs^nv$. Ainsi, $uXs^nv \to_g uXs^{n+1}v$ d'une part, et $uXs^nv \to_g urs^nv$ d'autre part. Comme la grammaire est $LL(k)$, nous avons $Pre_k(Xs^{n+1}v) \cap Pre_k(rs^nv) = \emptyset$. Comme $X \to r$, nous avons $Pre_k(rs^{n+1}v) \cap Pre_k(rs^nv) = \emptyset$. Si $s \to^* \varepsilon$, alors $Pre_k(rv) \subseteq Pre_k(rs^{n+1}v) \cap Pre_k(rs^nv)$. Or $Pre_k(rv) \neq \emptyset$ puisque la grammaire ne contient aucun non-terminal improductif. Donc, $s \not\to^* \varepsilon$. Mais, alors, il suffit de prendre $n > k$ pour aboutir à une contradiction. En effet, si $n > k$ alors, pour tout $r', s', v' \in \Sigma^*$ tel que $r \to^* r'$, $s \to^* s'$ et $v \to^* v'$, nous avons $(r'(s')^{n+1}v')|_k = (r'(s')^nv')|_k \in Pre_k(rs^{n+1}v) \cap Pre_k(rs^nv) = \emptyset$. ■

Afin d'essayer de transformer une grammaire non $LL(k)$ en une grammaire $LL(k)$ équivalente, ce qui est indécidable, il faut donc commencer par éliminer les préfixes communs et la récursivité.

Pour les préfixes communs, ce n'est pas difficile. Par exemple, la grammaire $\{X \to au \mid av\}$ est équivalente à la grammaire $\{X \to aX_a, X_a \to u \mid v\}$.

Pour la récursivité, nous avons:

Lemme 24 Toute grammaire propre est effectivement équivalente à une grammaire non récursive à gauche, c'est-à-dire, n'ayant aucun non-terminal X tel que $X \to^+ Xs$.

Preuve. Voyons d'abord comment éliminer la récursivité directe. Soit $G = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ une grammaire ayant pour un non-terminal X les règles $X \to r_1 \mid \ldots \mid r_k \mid Xs_1 \mid \ldots \mid Xs_l$ avec $l \geq 1$ et aucun X au début de $r_1, \ldots, r_l$. Comme la grammaire est propre, nous avons $s_i \neq \varepsilon$. Soit alors G' la grammaire équivalente non directement récursive à gauche qu'on obtient en remplaçant les règles précédentes par $X \to r_1X' \mid \ldots \mid r_kX'$ et $X' \to \varepsilon \mid s_1X' \mid \ldots \mid s_lX'$ où X' est un nouveau non-terminal.

Pour éliminer la récursivité générale, on suppose que $\mathcal{N} = \{X_1, \ldots, X_n\}$ et on applique l'algorithme suivant. Pour i allant de 1 à n faire:

- Pour j allant de 1 à $i-1$, si les règles de X_j sont $X_j \to r_1 \mid \ldots \mid r_k$, remplacer toute règle $X_i \to X_j s$ par les règles $X_i \to r_1 s \mid \ldots \mid r_k s$.
- Eliminer la récursivité directe à gauche de X_i. ■

Remarque: absence de préfixes communs, non-ambiguïté et non-récursivité ne sont pas des conditions suffisantes pour être $LL(k)$. Par exemple, $\{S \to aSb \mid A, A \to aA \mid \varepsilon\}$ est non-ambiguë, non-récursive et n'a pas de préfixes communs, mais n'est pas $LL(1)$ pour autant.

6.2 Cas particulier des grammaires $LL(1)$

Dans cette section, nous allons voir comment décider qu'une grammaire est $LL(1)$ et implémenter un algorithme d'analyse.

Si $L \neq \emptyset$, nous avons:

$$Pre_1(rL) = \begin{cases} (Pre_1(r) - \{\varepsilon\}) \cup Pre_1(L) \text{ si } r \to_g^* \varepsilon \\ Pre_1(r) \text{ sinon} \end{cases}$$

Lemme 25 Une grammaire réduite est $LL(1)$ ssi, pour toutes règles $X \to r_1 \neq X \to r_2$, $Pre_1(r_1 Suiv_1(X)) \cap Pre_1(r_2 Suiv_1(X)) = \emptyset$.

Preuve. Comme $Suiv_1(X) \subseteq \Sigma^*$, nous avons $Pre_1(Suiv_1(X)) = Suiv_1(X)$.

$\Rightarrow$ Par contradiction. Supposons que G soit $LL(1)$ et qu'il existe $X \to r_1 \neq X \to r_2$ et $w \in Pre_1(r_1 Suiv_1(X)) \cap Pre_1(r_2 Suiv_1(X))$.

- Cas $r_1 \not\to_g^* \varepsilon$ et $r_2 \not\to_g^* \varepsilon$. Alors, $w \in Pre_1(r_1) \cap Pre_1(r_2)$. Donc, il existe r'_1 et r'_2 tels que $r_1 \to_g^* wr'_1$ et $r_2 \to_g^* wr'_2$. Comme G est réduite, il existe u et v tel que $S \to_g^* uXv$. Donc il y a deux dérivations $S \to_g^* uXv \to_g ur_1v \to_g^* uwr'_1v'$ et $S \to_g^* uXv \to_g ur_2v \to_g^* uwr'_2v'$. Contradiction.
- Cas $r_1 \to_g^* \varepsilon$ et $r_2 \not\to_g^* \varepsilon$. Alors, $w \in Suiv_1(X) \cap Pre_1(r_2)$. Donc, il existe u, v et v' tels que $S \to_g^* uXv$, $v \to_g^* wv'$, et r'_2 tel que $r_2 \to_g^* wr'_2$. Donc il y a deux dérivations $S \to_g^* uXv \to_g ur_1v \to_g^* uv \to_g^* uwv'$ et $S \to_g^* uXv \to_g ur_2v \to_g^* uwr'_2v \to_g^* uwr'_2wv'$. Contradiction.
- Cas $r_1 \to_g^* \varepsilon$ et $r_2 \to_g^* \varepsilon$. Comme G est réduite, il existe u et v tels que $S \to_g^* uXv$. Donc il y a deux dérivations $S \to_g^* uXv \to_g ur_1v \to_g^* uv \to_g^* uv'$ et $S \to_g^* uXv \to_g ur_2v \to_g^* uv \to_g^* uv'$. Contradiction.

$\Leftarrow$ Par contraposée. Supposons que G ne soit pas $LL(1)$. Alors, il existe deux dérivations $S \to_g^* uXv \to_g ur_1v \to_g^* ur'_1v'_1 \in \Sigma^*$ et $S \to_g^* uXv \to_g ur_2v \to_g^* ur'_2v'_2 \in \Sigma^*$ telles que $(r'_1v'_1)|_1 = (r'_2v'_2)|_1$ et $r_1 \neq r_2$. Si $r'_i = \varepsilon$, alors $(r'_iv'_i)|_1 = (v'_i)|_1 \in Suiv_1(X) = Pre_1(Suiv_1(X)) \subseteq Pre_1(r_i Suiv_1(X))$ car $Suiv_1(X) \neq \emptyset$. Sinon, $(r'_iv'_i)|_1 = (r'_i)|_1 \in Pre_1(r_i) \subseteq Pre_1(r_i Suiv_1(X))$. Dans les deux cas, $Pre_1(r_1 Suiv_1(X)) \cap Pre_1(r_2 Suiv_1(X)) \neq \emptyset$. ■

Corollaire 3 Une grammaire réduite est $LL(1)$ ssi, pour toutes règles $X \to r_1 \neq X \to r_2$, $Pre_1(r_1) \cap Pre_1(r_2) = \emptyset$ et, si $r_1 \to_g^* \varepsilon$, $Suiv_1(X) \cap Pre_1(r_2) = \emptyset$.

Corollaire 4 Etant donnée une grammaire $LL(1)$ réduite, pour tout (X, a), il existe au plus une règle $X \to r \in \mathcal{R}$ tel que $a \in Pre_1(rSuiv_1(X))$.

Maintenant, comment calculer $Pre_1(r)$?

Soit $E = \mathcal{N} \to \wp(\Sigma \cup \{\varepsilon\})$ l'ensemble des fonctions de $\mathcal{N}$ dans les parties de $\Sigma \cup \{\varepsilon\}$. L'ensemble E est fini car $\mathcal{N}$ et Σ sont finis. Soit $\leq_E$ l'ordre sur E tel que $\mathcal{Y} \leq_E \mathcal{Z}$ ssi, pour tout $X \in \mathcal{N}$, $\mathcal{Y}(X) \subseteq \mathcal{Z}(X)$. $(E, \leq_E)$ est un treillis complet.

Soit $\overline{E} = (\mathcal{N} \cup \Sigma)^* \to \wp(\Sigma \cup \{\varepsilon\})$ l'ensemble des fonctions de l'ensemble des mots sur $\mathcal{N} \cup \Sigma$ dans les parties de $\Sigma \cup \{\varepsilon\}$. Soit $\leq_{\overline{E}}$ l'ordre sur $\overline{E}$ tel que $\mathcal{Y} \leq_{\overline{E}} \mathcal{Z}$ ssi, pour tout $v \in (\mathcal{N} \cup \Sigma)^*$, $\mathcal{Y}(v) \subseteq \mathcal{Z}(v)$. $(\overline{E}, \leq_{\overline{E}})$ est un treillis complet.

Etant donnée une fonction $\mathcal{Y}$ de E, soit $\overline{\mathcal{Y}}$ son extension à $\overline{E}$ définie par récurrence sur la longueur de $v \in (\mathcal{N} \cup \Sigma)^*$:

- $\overline{\mathcal{Y}}(\varepsilon) = \{\varepsilon\}$
- $\overline{\mathcal{Y}}(au) = \{a\}$
- $\overline{\mathcal{Y}}(Xu) = \mathcal{Y}(X)$ si $\varepsilon \notin \mathcal{Y}(X)$
- $\overline{\mathcal{Y}}(Xu) = \mathcal{Y}(X) \cup \overline{\mathcal{Y}}(u)$ si $\varepsilon \in \mathcal{Y}(X)$

Nous avons $Pre_1(r) = \overline{P}(r)$ où $P(X) = \{v|_1 \mid \exists v \in \Sigma^*, X \to^* v\}$.

Maintenant, remarquons que $P = F(P)$ où $F(\mathcal{Y})(X) = \bigcup\{\overline{\mathcal{Y}}(r) \mid X \to r \in \mathcal{R}\}$. On vérifie que F est monotone. Cela découle du fait que, si $\mathcal{Y} \leq_E \mathcal{Z}$, alors $\overline{\mathcal{Y}} \leq_{\overline{E}} \overline{\mathcal{Z}}$. Ainsi, d'après le théorème de Tarski (voir cours 1), l'équation $\mathcal{Y} = F(\mathcal{Y})$ admet une plus petite solution $A \subseteq P$ calculable par itération de F à partir de $\emptyset$. On prouve ensuite que $P \subseteq A$.

Enfin, comment calculer $Suiv_1(X)$?

Remarquons que $Suiv_1 = G(Suiv_1)$ où G est la fonction sur E telle que $G(\mathcal{Y})(X) = \bigcup\{H(\mathcal{Y}, Z, X, v) \mid Z \to uXv \in \mathcal{R}\}$ où:

- $H(\mathcal{Y}, Z, X, v) = (Pre_1(v) - \{\varepsilon\}) \cup \mathcal{Y}(Z)$ si $\varepsilon \in Pre_1(v)$
- $H(\mathcal{Y}, Z, X, v) = Pre_1(v)$ si $\varepsilon \notin Pre_1(v)$

On vérifie que G est monotone. Ainsi, d'après le théorème de Tarski, l'équation $\mathcal{Y} = G(\mathcal{Y})$ admet une plus petite solution $B \subseteq Suiv_1$ calculable par itération de F à partir de $\emptyset$. On prouve ensuite que $Suiv_1 \subseteq B$.

6.3 Analyse syntaxique $LL(1)$: automates à pile

Maintenant, étant donnée une grammaire $LL(1)$ G, nous pouvons donner un algorithme pour savoir si un mot u appartient à $L(G)$. Pour cela, nous commençons par construire un tableau à 2 dimensions T qui, à chaque paire $(X, a) \in \mathcal{N} \times (\Sigma \cup \{\varepsilon\})$, associe le membre droit r de l'unique règle $X \to r \in \mathcal{R}$ (si elle existe) telle que $a \in Pre_1(rSuiv_1(X))$. Alors, on peut décider si $u \in L(G)$ en utilisant la procédure suivante:

```
main() =
   Soit p un pointeur sur la première lettre de u$
      où $ est un nouveau caractère marquant la fin de u.
   Appeler S().
   Si p ne pointe pas sur $ alors erreur.
```

où, pour chaque non-terminal X, nous avons une procédure:

```
X() =
   Soit a la lettre vers laquelle pointe p.
   Soit a' = (si a = $ alors ε sinon a).
   Si T(X,a') n'est pas défini, alors erreur.
   Sinon, soit b_1...b_n la valeur de T(X,a').
   Pour i allant de 1 à n, faire:
      Si b_i ∈ Σ, alors consommer(b_i), sinon appeler b_i()
```

où:

```
consommer(b) = si p pointe sur b alors avancer p sinon erreur
```

Alternativement, on peut utiliser une machine abstraite utilisant une pile. On appelle configuration une paire (v, u) où $u \in \Sigma^*$ est un mot à reconnaître à partir de $v \in (\Sigma \cup \mathcal{N})^*$ (la pile). La machine peut alors faire les transitions suivantes selon les 1-préfixes de la pile et du mot à lire:

- $(Xv, au) \to (rv, au)$ si $T(X, a) = r$ (on remplace X par r),
- $(Xv, \varepsilon) \to (rv, \varepsilon)$ si $T(X, \varepsilon) = r$ (idem),
- $(av, au) \to (v, u)$ (on consomme a),
- $(v, u) \to \perp$ (erreur) dans tous les autres cas où $v \neq \varepsilon$ et $u \neq \varepsilon$.

On peut alors montrer que $(v, u) \to^* (v', u')$ ssi il existe q tel que $u = qu'$ et $v \to_g^* qv'$. Ainsi, $u \in L(G)$ ssi $(S, u) \to^* (\varepsilon, \varepsilon)$. De plus, à chaque étape, une et une seule transition est possible (la machine est déterministe).

Exemple 1 Prenons la grammaire $S \to AB, A \to aAb\varepsilon, B \to bB \mid \varepsilon$.

Remarquons tout d'abord que B et A sont effaçables (règles $B \to \varepsilon$ et $A \to \varepsilon$), et donc S aussi (règle $S \to AB$).

Pour remplir la table, il nous faut calculer, pour chaque règle $X \to r$, $Pre_1(rSuiv_1(X))$:

- $S \to AB$:
  - $Pre_1(ABSuiv_1(S)) = (Pre_1(A) - \{\varepsilon\}) \cup (Pre_1(B) - \{\varepsilon\}) \cup Suiv_1(S)$
  - $Pre_1(A) = \{a, \varepsilon\}$ car $A \to aAb$ ou $A \to \varepsilon$
  - $Pre_1(B) = \{b, \varepsilon\}$
  - $Suiv_1(S) = Pre_1(\widehat{Suiv}(S))$
  - $\widehat{Suiv}(S) = \{\varepsilon\}$ car S n'apparaît dans aucun membre de règle, donc
  - $Suiv_1(S) = \{\varepsilon\}$ et
  - $Pre_1(ABSuiv_1(S)) = \{a, b, \varepsilon\}$, ce qui nous permet de remplir par AB la première ligne du tableau, c'est-à-dire les cases (S, a), (S, b) et (S, ε)
- $A \to aAb$:
  - $Pre_1(aAbSuiv_1(A)) = \{a\}$, ce qui nous permet de remplir la case (A, a) avec aAb
- $A \to \varepsilon$:
  - $Pre_1(\varepsilon Suiv_1(A)) = Suiv_1(A) = Pre_1(\widehat{Suiv}(A))$
  - $\widehat{Suiv}(A) = \{B, b\}$, donc
  - $Pre_1(\varepsilon Suiv_1(A)) = Pre_1(B) \cup Pre_1(b) = \{b, \varepsilon\}$, ce qui nous permet de remplir les cases (A, b) et (A, ε) avec ε
- $B \to bB$:
  - $Pre_1(bBSuiv_1(B)) = \{b\}$, ce qui nous permet de remplir la case (B, b)
- $B \to \varepsilon$:
  - $Pre_1(\varepsilon Suiv_1(B)) = Suiv_1(B) = Pre_1(\widehat{Suiv}(B))$
  - $\widehat{Suiv}(B) = \{\varepsilon\}$, donc
  - $Pre_1(\varepsilon Suiv_1(B)) = \{\varepsilon\}\}$, ce qui nous permet de remplir par ε la case (B, ε)

	a	b	ε
S	AB	AB	AB
A	aAb	ε	ε
B		bB	ε

On remarque qu'il y au plus une règle par case: la grammaire est $LL(1)$, et que la case (B, a) est vide: si, dans l'état B, le premier caractère est a, c'est une erreur de syntaxe. ■