CHAPITRE

DÉFINITIONS ENSEMBLISTES DES LANGAGES, MOTS ET OPÉRATIONS

2

Nous allons voir dans ce chapitre les bases fondamentales pour définir et étudier les langages formels d'un point de vue ensembliste (définition par extension, par compréhension et par induction). Nous nous intéressons aussi à la brique élémentaire qui constitue un langage (un mot) et les opérations qui viennent avec, et qui seront généralisées par la suite sur les langages avec d'autres opérations spécifiques

2.1 Notions de base

Définition 2.1.1 Un langage formel

Un langage formel L est un ensemble (fini ou infini) de mots définis sur un alphabet X,

Exemple

- *Le langage des nombre binaires (infini) définis sur l'alphabet* $\{0, 1\}$.
- *Le langage :* $\{ac, abc\}$ *ce sont des mots définis sur* $\{a, b, c\}$ *commençant par a et se terminant avec c (C'est un langage fini).*
- *Le langage C (mais sur quel alphabet ?).*

Définition 2.1.2 Alphabet

Un alphabet est un ensemble de symboles, appelé également vocabulaire.

Définition 2.1.3 Symbole

Un symbole est une entité abstraite. Par exemple : des lettres, des chiffres, des symboles graphiques, et d'autres signaux (idéogrammes, signal lumineux, des sons, fumée, etc).

Définition 2.1.4 Mot

Un mot défini sur un alphabet X est une séquence (une chaîne) finie de symboles juxtaposés de X.

Exemple

— *Le mot 1011 est défini sur l'alphabet* $\{0, 1\}$.

— *Le mot 1.23 est défini sur l'alphabet* $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, .\}$

— ε *représente le mot vide.*

2.2 Définitions ensemblistes d'un langage

Définition 2.2.1 Un ensemble

Un ensemble est une collection d'objets sans répétition, chaque objet est appelé un élément. L'ensemble vide, noté $\emptyset$ est un ensemble qui ne contient aucun élément. Un élément dans un ensemble représentant un langage est un mot.

La définition ensembliste d'un langage L est utile lorsqu'on s'intéresse à démontrer si une propriété Q est vérifiée par tous les mots de l'ensemble L. Mais il faut noter qu'un ensemble peut être défini selon plusieurs façons, chacune a ses avantages et ses inconvénients. Par conséquent, la démonstration de Q sur L dépend de sa définition ensembliste adoptée.

Beaucoup de notions relatives à la théorie des ensembles seront définies tout en faisant le parallèle avec la logique mathématique (logique des prédicats).

2.2.1 Définition par extension

Définition 2.2.2

Définir un ensemble (un langage) L par extension consiste à lister tous ses éléments (l'ordre n'est pas important). Si un élément w figure dans l'ensemble L, alors on écrit $w \in L$.

Exemple

Le langage des entiers < 10 *est représenté par l'ensemble* $L = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Remarque 2.2.1

La définition par extension est faisable que lorsque le nombre d'éléments d'un ensemble est fini et n'est pas très important. Même s'il est possible de noter par ... certains ensembles infinis, mais cette définition n'est pas représentative ou pratique.

Montrer si une propriété Q est vérifiable par L

Une propriété Q est vérifiée par L si elle l'est pour tout élément de L : C'est à dire $\forall w \in L, Q(w)$ est vraie. Cette démonstration ne peut fonctionner que si l'ensemble L est fini et elle n'est pas pratique si le nombre d'éléments est important dans un langage défini par extension.

2.2.2 Définition par compréhension

Définition 2.2.3 Par compréhension

Définir un langage L par compréhension consiste à définir un prédicat P qui ne peut prendre la valeur "vraie" que si, et seulement si, il est vérfié par tous les éléments de L, $L = \{w | P(w)\}$.

Remarque 2.2.2

La définition par compréhension est plus pratique car elle permet de définir aisément des ensembles (des langages) ayant un nombre infini d'éléments (de mots).

Exemple

L'ensemble des entiers naturels multiples de 3 est défini par : $\{w \in N | w \bmod 3 = 0\}$ *(mod est l'opération modulo).*

Montrer si une propriété Q est vérifiable par L

Dans le cas de cette définition, Q est vérifiée par L si on arrive à démontrer que $P(w) \rightarrow Q(w)$ en utilisant les axiomes de l'ensemble en question. Cependant, la question se pose sur la complétude et la démontrabilité des théorèmes. Quelque soit l'ensemble des axiomes définissant les entiers, il existe des propriétés des nombres entiers (des propriétés vraies) dans ce système formel (l'arithmétique des entiers) mais qui ne peuvent pas être démontrés à l'intérieur de ce système.

2.2.3 Définition par induction

Définition 2.2.4

Définir un langage L par induction consiste à montrer comment les éléments de L sont construits. Cette définition nécessite de définir :

- Un ensemble fini d'éléments triviaux de L, noté $triv(L)$ dont on suppose l'appartenance à L, et qui constituent la base de la définition inductive (point de départ).
- Règles inductives permettant de générer de nouveaux éléments à partir des éléments déjà définis. Il s'agit d'une ou plusieurs fonctions $f_i : L \rightarrow L$: des règles de la forme : $w \rightarrow f(w)$.

Donc : $L = \{triv(L); w \in L \rightarrow f_i(w) \in L\}$

Remarque 2.2.3

— Cette définition est une preuve d'appartenance pour tout élément de L.

— La définition par induction, tout comme la définition par compréhension, permet de définir des ensembles infinis.

— Elle peut simplifier la définition de certains ensembles complexes, ou des structures mathématiques en définissant ses éléments ou ses propriétés de manière progressive et avec une meilleure structuration.

Exemple

On peut définir les entiers naturels par induction : $N = \{\{0\}; x \in N \rightarrow x + 1 \in N\}$

Montrer si une propriété Q est vérifiable par L (démonstration par induction)

Avec cette définition inductive, pour montrer qu'une propriété Q est vérifiée par L, il faut :

1. Montrer que tout élément de $triv(L)$ vérifie Q. Cette vérification est faisable car l'ensemble des triviaux est généralement de petite taille.
2. Supposer que w satisfait Q, et démontrer que $f(w)$ satisfait également Q : $Q(w) \rightarrow Q(f(w))$ Cette méthode de démonstration appelée une démonstration par induction est très utile pour démontrer des propriétés très complexes.

Exemple

Considérons l'ensemble des entiers naturels N qui a été défini précédemment par induction. Pour démontrer la satisfaction d'une propriété Q sur N, il faut :

1. *Vérifier $Q(0)$ (Base d'induction).*
2. *Pour $x \in N$, supposer que $Q(x)$ est vraie ou vérifiée (Hypothèse d'induction) et démontrer que $Q(x+1)$ est vérifiée.*

Lorsque ce type de raisonnement s'applique sur les entiers, on dit que c'est **un raisonnement par récurrence**. Autrement dit, le raisonnement par récurrence n'est qu'un cas particulier du raisonnement par induction.

Déduction 2.2.1 $w \in L$ d'un point de vue ensembliste

Pour répondre à la question d'appartenance de $w \in L$ d'un point de vue ensembliste, il existe plusieurs façons pour y répondre :

- On vérifie soit l'existence de w dans la liste des mots de L (impossible à réaliser si le langage est infini).
- Montrer que w vérifie une propriété P celle qui définit L.
- Montrer comment obtenir (produire) w selon la définition inductive de L.

2.3 Notions et Opérations sur les mots

2.3.1 Longueur d'un mot

Définition 2.3.1 La longueure $|w|$

- La longueur d'un mot w (défini sur X) notée $|w|$ est définie comme étant le nombre de symboles qui forment w. Si $|w| = 0$ alors $w = \varepsilon$.
- Si on définit la longueur par induction :
  - $|\varepsilon| = 0$.
  - $|aw| = |w| + 1, a \in X$.

Exemple

- $|ab| = 2, |abcc| = 4$.
- *Pratiquement, pour mieux comprendre, on peut illustrer un mot $w = ab$ par une chaîne de caractères (par exemple en langage Python). "ab". Sa longueur est donnée par $len(\text{"}ab\text{"}) = 2$*
- *Si $w = \varepsilon$ alors w est représenté par la chaîne vide "", sachant que $|\varepsilon| = 0$ c'est l'équivalent de $len(\text{""}) = 0$.*

2.3.2 Nombre d'occurrences d'un symbole dans un mot

Définition 2.3.2

Le nombre d'occurrences d'un symbole a dans un mot w est noté par $|w|_a$.

Exemple

$|abba|_a = 2$.

Remarque 2.3.1

Il est possible d'avoir des mots de longueur infinie. Mais tous les mots qu'on manipule dans ce cours sont considérés de longueur finie.

2.3.3 Concaténation des mots

Définition 2.3.3 Concaténation

Soient $v = a_1a_2 \dots a_n$ et $u = b_1b_2 \dots b_n$ deux mots définis sur l'alphabet X. La concaténation de v et u est obtenue en écrivant u suivi de v, ce qui donne un mot $w = a_1a_2 \dots a_nb_1b_2 \dots b_n$ tel que w est également défini sur X.

Propriétés de la concaténation

Soient u, v, w trois mots définis sur l'alphabet X :

- La concaténation n'est pas commutative $uv \neq vu$, $(u \neq v)$
- Associativité : $(u.v).w = u.(v.w)$;
- $u.\varepsilon = \varepsilon.u = u$, ($\varepsilon$ est un élément neutre pour la concaténation)
- Fermeture : Si u et v sont définis sur l'alphabet X alors $u.v$ est également défini sur X
- $\forall a \in X, |a.u| = |u| + 1$
- $|u.v| = |u| + |v|$
- $\forall a \in X, |u.v|_a = |u|_a + |v|_a$

Remarque 2.3.2

- La concaténation peut être notée par le point, mais il peut être omis s'il n'y a pas d'ambiguïté : $w = v.u = vu$.
- L'ensemble de tous les mots définis sur X, noté par exemple A, forme avec la concaténation un monoïde noté $(A, ., \varepsilon)$. C'est une structure algébrique qui possède des propriétés (Fermeture, Associativité, et élément neutre) pouvant être utilisées dans l'analyse des langages.

Théorème 2.3.1

De $(A, ., \varepsilon)$ vers $(N, +, 0)$, la longeur est une application homomorphisme car :

- $(A, ., \varepsilon)$ et $(N, +, 0)$, sont deux monoïdes.
- $|u.v| = |u| + |v|$

2.3.4 L'exposant

Définition 2.3.4 Exposant (Puissance)

L'opération exposant $w^n = \underbrace{w \dots w}_{\text{n fois}}$. Sa définition par induction est donnée par :

- $w^0 = \varepsilon$
- $w^1 = w$

— $w^{n+1} = w^n.w$

2.3.5 Le miroir d'un mot

Définition 2.3.5 Mot miroir

Soit $w = a_1a_2 \dots a_n$ un mot défini sur X ($a_i \in X$). Le miroir de w est obtenu en écrivant w à l'envers et cela est noté par : $w^R = a_n \dots a_2a_1$. On peut donner aussi une définition inductive de w^R :

— $\varepsilon^R = \varepsilon$

— $a^R = a$

— $(aw)^R = w^R.a, a \in X$

Propriétés

— $(w^R)^R = w$.

— $(u.v)^R = v^R.u^R$.

— Lorsque $w^R = w$, w est appelé un mot palindrome qui permet de lire la même chose dans les deux directions

Définition 2.3.6 Le mot palindrome

Tout mot palindrome défini sur l'alphabet X est un mot qui s'écrit sous la forme ww^R (de longueur paire) ou waw^R (de longueur impaire), tel que $a \in X$.

2.3.6 Préfixe et suffixe

Définition 2.3.7

On appelle p (respectivement s) le préfixe (respectivement le suffixe) d'un mot w défini sur X, si $w = pu$ (respectivement $w = vs$). v et u sont deux mots définis sur X, sachant que p (respectivement s) est formé par 0 ou plusieurs symboles de X.

D'une manière générale, si $w = a_1a_2 \dots a_n$,

— l'ensemble des préfixes de w $Pref(w) = \{\varepsilon, a_1, a_1a_2, a_1a_2a_3, a_1a_2 \dots a_n\}$.

— l'ensemble des suffixes de w $Suf(w) = \{\varepsilon, a_n, a_{n-1}a_n, a_{n-2}a_{n-1}a_n, a_1a_2 \dots a_{n-2}a_{n-1}a_n, \}$.

2.3.7 Factorisation

Définition 2.3.8 Facteur

On dit que v est un facteur (substring) de w si et seulement s'il existe deux mots x et y, tel que : w = xuy (w, v, x, y sont définis sur X).

Remarque 2.3.3

- On dit que u est un facteur propre de w si et seulement si $x \neq \varepsilon$ et $y \neq \varepsilon$.
- si $x = \varepsilon$ alors u est appelé un facteur gauche de w. Dans ce cas là, $u \in Pref(w)$
- si $y = \varepsilon$ alors u est appelé un facteur droit de w. Dans ce cas là, $u \in Suf(w)$.

Remarque 2.3.4 Sous-mot

- Un facteur peut désinger aussi un sous-mot. Mais contrairement aux définitions précédentes, un sous-mot est copmposé de fragments non nécessairement contigus, mais dans lesquels l'ordre d'apparition des symboles est toutefois respecté.
- Il faut faire attention à l'équivalence entre la terminologie française et anglaise. Un facteur est appelé en anglais : 'subword' ou 'substring'. Alors que 'subsequence' ou 'scattered subword' est l'équivalent en anglais de sous-mot.

Déduction 2.3.1

Si $Fact(w)$ ou $Substring(w)$ est l'ensemble des facteurs de w, alors on peut déduire que $Pref(w) \subset Fact(w)$ (resp. $Suf(w) \subset Fact(w)$)

2.3.8 Entrelacement

Définition 2.3.9 Entrelacement (Shuffle)

Soit $u = u_1u_2 \dots u_n$ et $v = v_1v_2 \dots v_m$ deux mots. L'entrelacement ou le mélange de u et v noté par $u \sqcup\!\sqcup v$ est défini par l'ensemble des mots w qui représente un mélange entre les symboles de u de v. tout en gardant l'ordre d'apparition des symboles dans u et dans v.

Remarque 2.3.5

— L'entrelacement est commutatif.

— lorsque $v = a$, l'entrelacement de $u \sqcup\!\sqcup a = \{u_1.a.u_2 | u = u_1.u_2\}$, tel que $a \in X$ et cela consiste à insérer le symbole a à n'importe quelle position dans u.

Exemple

— $ab \sqcup\!\sqcup c = \{cab, acb, abc\}$

— $ab \sqcup\!\sqcup ac = \{abac, aabc, aacb, acab\}$

Cette liste d'opérations sur les mots n'est pas exhaustive, car il existe d'autres opérations : Distances entre mots, Quotient, Ordre sur les mots, etc.

2.4 Opérations sur les langages

Soient L_1 et L_2 deux langages formels quelconques, considérés comme deux sous-ensembles d'un ensemble plus grand Ω.

2.4.1 Opérations ensemblistes

Selon les définitions ensemblistes que nous avons précédemment données aux langages formels, on peut définir L_1 et L_2 :

— Par compréhension : $L_1 = \{u | P(u)\}$ et $L_2 = \{v | Q(v)\}$.

— Par induction : $L_1 = \{triv(L_1); u \to f(u)\}$ et $L_2 = \{triv(L_2); v \to g(v)\}$.

Par conséquent, plusieurs opérations ensemblistes sont également applicables.

Inclusion

$L_1 \subseteq L_2$ veut dire que :

— $\forall w \in L_1 \rightarrow w \in L_2$.

— $P \rightarrow Q$ dans le cas de la définition par compréhension.

Remarque 2.4.1

— $L_1 \subseteq L_2$ signifie que L_1 est un sous-ensemble (et pas un sous-langage) de L_2.

— $\emptyset \subseteq L_1$, peu importe L_1

— $L_1 \subseteq L_2$ et $L_2 \subseteq L_1 \Rightarrow L_1 = L_2$

Union

L'union entre les ensembles est notée par $\cup$. Mais quand il s'agit de l'union entre langages on utilisera plutôt $+$ ou $|$. $L_1 \cup L_2 = L_1 + L_2 = L_1|L_2$

— $L_1 + L_2 = \{w|w \in L_1 \vee w \in L_2\}$.

— Par compréhension : $L_1 + L_2 = \{w|P(w) \vee Q(w)\}$

— Par induction : $L_1 + L_2 = \{triv(L_1) \cup triv(L_2); w \rightarrow f(w), w \rightarrow g(w)\}$

Intersection

— $L_1 \cap L_2 = \{w|w \in L_1 \wedge w \in L_2\}$

— Par compréhension : $L_1 \cap L_2 = \{w|P(w) \wedge Q(w)\}$

— D'une manière générale il n y a pas de définition inductive donnée à l'intersection

Différence

— $L_1 - L_2 = \{w|w \in L_1 \wedge w \notin L_2\}$

— Par compréhension : $L_1 - L_2 = \{w|P(w) \wedge \neg Q(w)\}$

— D'une manière générale il n y a pas de définition inductive donnée à la différence.

Complément

— $\overline{L_1} = \Omega - L_1 = \{w|w \notin L_1 \wedge w \in \Omega\}$

— Par compréhension : $\overline{L_1} = \{w|\neg P(w)\}$

— D'une manière générale il n y a pas de définition inductive donnée à au complément.

Produit cartésien

— $L_1 \times L_2 = \{(u, v)|u \in L_1 \wedge v \in L_2\}$

— Par compréhension : $L_1 \times L_2 = \{(u, v)|P(u) \wedge Q(v)\}$

— Par induction : $L_1 \times L_2 = \{triv(L_1) \times triv(L_2); (u, v) \rightarrow (f(u), g(v))\}$

Cardinalité

La cardinalité $card(L_1)$ de n'importe quel ensemble L_1 représente le nombre de ses éléments. Par exemple : $card(L_1 \times L_2) = card(L_1) \times card(L_2)$

Ensemble des parties

L'ensemble des parties 2^L d'un ensemble L c'est l'ensemble de tous les sous-ensembles de L. On peut déduire que $card(2^L) = 2^{card(L)}$.

Exemple

Soit $L_1 = \{aa, bb\}, L_2 = \{aa, cc\}, \Omega = \{aa, ab, bb, cc\}$,

- $L_1 \cap L_2 = \{aa\}$.
- $L_1 \cup L_2 = \{aa, bb, cc\}$.
- $L_1 - L_2 = \{bb\}$.
- $\overline{L_1} = \{ab, cc\}$.
- $L_1 \times L_2 = \{(aa, aa), (aa, cc), (bb, aa), (bb, cc)\}$
- $2^{L_1} = \{\emptyset, \{aa\}, \{bb\}, \{aa, bb\}\}$

2.4.2 Autres opérations spécifiques

Soient L, L_1, L_2 et L_3 des langages définis sur l'alphabet X.

Concaténation

$L_1.L_2 = \{w | \exists u \in L_1 \wedge \exists v \in L_2 : w = uv\}$ ou encore : $L_1.L_2 = \{w | \exists (u, v) \in L_1 \times L_2 : w = uv\}$

Remarque 2.4.2 Propriétés de la concaténation

- $L.L \neq L$
- $L_1.L_2 \neq L_2.L_1$
- $(L_1.L_2).L_3 = L_1.(L_2.L_3)$ (Associativité comme avec les mots).
- $L_1.(L_2 + L_3) = (L_1.L_2) + (L_1.L_3)$. Distributivité de la concaténation sur l'union
- $L_1.(L_2 \cap L_3) \neq (L_1.L_2) \cap (L_1.L_3)$. La concaténation n'est pas distributive sur l'intersection
- $L_1.(L_2 \cap L_3) \subseteq (L_1 \cap L_2).(L_1 \cap L_3)$

Préfixes d'un langage

Les préfixes d'un langage L sont définis par l'ensemble des préfixes de chaque mot de L.
$Pref(L) = \{u | \exists w \in L : u \in Pref(w)\}$. On l'appelle aussi le langage des préfixes (à ne pas confondre avec le langage préfixe)

Suffixes d'un langage

Les suffixes d'un langage L sont définis par l'ensemble des suffixes de chaque mot de L.
$Suf(L) = \{u | \exists w \in L : u \in Suf(w)\}$. On l'appelle aussi le langage des suffixes.

Exposant

L'exposant (ou la puissance concaténative) d'un langage :
$L^n = \underbrace{L.L \ldots L}_{\text{n fois}} = \{w | \exists u_1, u_2, \ldots u_n \in L_1 : w = u_1 u_2 \ldots u_n, n \geq 0\}$.
On peut également donner une définition inductive :

- $L^0 = \{\varepsilon\}$.
- $L^1 = L$.
- $L^n = L.L^{n-1}$

Fermeture transitive de Kleene

Définition 2.4.1

La fermeture transitive (la fermeture itérative ou l'étoile) de Kleene est définie par : $L^* = \sum_{i \geqslant 0} L^i$.
Une autre définition possible :
$L^* = \{w | \exists n \geqslant 0, \exists u_1 \in L, \exists u_2 \in L, \ldots \exists u_n \in L, w = u_1 u_2 \ldots u_n\}$.

Donc L^* est l'ensemble de tous les mots que l'on peut construire en concaténant un nombre fini (éventuellement réduit à zéro) d'éléments du langage L.
Si $L = X$ (l'alphabet), alors X^* est l'ensemble de tous les mots possibles sur X.Autrement dit, un langage L défini sur X est un sous-ensemble de X^*.

Fermeture non transitive

La fermeture non transitive (la fermeture itérative propre ou l'étoile propre) de Kleene est définie par : $L^+ = \sum_{i>0} L^i$

Remarque 2.4.3 Propriétés de la fermeture de Kleene

- $\emptyset^* = \varepsilon$
- $L^* = L^+ + \{\varepsilon\}$ (L^+ ne contient ε que si $\varepsilon \in L$)
- $L^+ = L.L^*$
- $(L^*)^* = L^*$
- $L^*.L^* = L^*$
- $L_1.(L_2.L_1)^* = (L_1.L_2)^*.L_1$
- $(L_1 + L_2)^* = (L_1{}^*.L_2{}^*)^*$
- $(L_1 + L_2)^* \neq L_1{}^* + L_2{}^*$

Ce genre d'opérateurs et propriétés permettent d'exprimer de manière formelle (et compacte) des langages complexes, éventuellement infinis, à partir des langages plus simples.

Langage miroir

$L^R = \{w | \exists u \in L : w = u^R\}$

Entrelacement ou mélange de langages

- $L_1 \sqcup\!\sqcup L_2 = \{w | \exists u \in L_1 \wedge \exists v \in L_2 : w \in u \sqcup\!\sqcup v\}$.
- L'entrelacement entre un langage L et un symbole a est l'union des ensembles résultants de l'entrelacement de chaque mot de L avec a. Plus simplement : $L \sqcup\!\sqcup a = \{w \sqcup\!\sqcup a = u.a.v | w = u.v \in L\}$

Remarque 2.4.4 Opérations régulières

L'union, l'étoile de Kleen, la concaténation, le miroir sont appelées aussi des opérations régulières (On en parlera dans le chapitre suivant)

2.5 Série d'exercices de TD N°1

Exercice 1 : L'alphabet d'un langage formel

Trouvez l'alphabet de chacun des langages suivants :

1. *Les nombres binaires.*
2. *Les nombres entiers éventuellement munis d'un signe.*
3. *Le langage contenant les mots* $\{\varepsilon, +, a+b, a+10, 00\}$
4. *Les nombres réels en C.*
5. *Les identifiants en C.*
6. *Le langage C.*

Exercice 2 : Définitions ensemblistes

Pour chacun des langages suivants, donnez une definition formelle correspondante à sa description et précisez le type de cette définition.

1. *Tous les mots sur* $\{a, b, c\}$ *de longueur* 2 *et ne contenant pas un* c.
2. *Tous les mots sur* $\{a, b\}$ *contenant au maximum deux* a *ou bien un* b.
3. *Tous les mots sur* $\{a, b\}$ *contenant plus de* a *que de* b.
4. *Le langage des mots* L *contenant* ε*, et si* $u \in L$ *alors* $auab \in L$.
5. *Les entiers multiples de 11 s'écrivant sous la forme* $10^{2n}1$.
6. *Les entiers palindromes de longueur paire.*

Exercice 3 : Opération sur les langages (opérations ensemblistes, concaténation)

Calculez :

1. *Le complément de l'ensemble des mots définis sur* $\{0, 1\}$ *commençant par* 1.
2. $\{a^i b^j | i, j \geq 0\} \cap \{ab, b\}$
3. $\{a^i b^j | i, j \geq 0\} \cup \{aa\}$
4. $\{a, bc\}.\{bb, c\}$
5. $\{a^n | n > 0\}.\{b^m | m > 0\}$

Exercice 4 : Opérations sur les mots/langages (Entrelacement)

1. *Calculez :*
   (a) $\varepsilon \sqcup\!\sqcup\, a$.
   (b) $abca \sqcup\!\sqcup\, d$.
   (c) $abca \sqcup\!\sqcup\, a$.
   (d) $a^n \sqcup\!\sqcup\, b.\ n \geq 0$
2. *Calculez :*
   (a) $\{a^n | n \geq 0\} \sqcup\!\sqcup\, a$.
   (b) $\{a^n b^n | n \geq 0\} \sqcup\!\sqcup\, a$

Exercice 5 : Opérations sur les mots/langages (Préfixes/Suffixes)

1. *Calculez les préfixes* $Pref(L)$ *des langages suivants :*
   (a) $L_1 = \{ab, abc, \varepsilon\}$.
   (b) $L_2 = \{a^n b^m | n, m \geq 0\}$.

(c) $L_3 = \{a^n b^n | n \geq 0\}$.

2. *Calculez les Suffixes $Suf(L)$ des langages cités ci-dessus.*
3. *Calculez les facteurs propres de : $L_1, \{while\}$*

Exercice 6 : Opérations sur les langages (L'étoile de Kleen)

Calculez la fermeture de Kleene de chacun des langages suivants et donnez une définition formelle ou une description possible du résultat :

1. $L_1 = \{\varepsilon\}$.
2. $L_2 = \{a\}$.
3. $L_3 = \{a, ab\}$.
4. $L_4 = \{aa, ab, ba, bb\}$.

Exercice 7 : Définir un langage en raisonnant sur la longueur de ses mots

1. *Trouvez le langage L défini sur X tel que $L = \{w \in X^* | w^2 = w^3\}$*
2. *Trouvez le langage L défini sur X tel que $L = \{w \in X^* | \exists v \in X^* : w^3 = v^2\}$*

2.6 Série de TP N°1

L'objectif de cette première série d'exercices de travaux pratiques est de pouvoir mettre en pratique les notions qui ont été abordées dans ce chapitre : Définir des langages (définition ensembliste), et manipuler des mots et des langages, etc. Pour cela, nous aurons besoins de :

Prérequis

- *Les notions vues en cours/TD liées à ce chapitre*
- *Langage de programmation Python*
- *Des rappels sur quelques notions liées à la manipulation des chaînes de caractères, des listes (et éventuellement les ensembles) en Python. En revanche, ces petits rappels ne remplacerons pas la documentation officielle.*

Quelques notions sur: Les listes

- Une liste est une collection ordonnée d'éléments, modifiable et de n'importe quel type de données (même hétérogènes et avec répétition). Contrairement à un Set qui est une collection d'éléments qui ne peuvent pas être modifiés. Le tableau suivant donne quelques exemples de manipulation sur les listes.

La manipulation	Code	Exécution
Définir une liste	L=[1,2,5,6]	
Créer une liste avec range	list(range(0,4))	[0,1,2,3]
Une liste de listes	[1,2,[3,4],4.5]	
Nbr d'd'éléments (Cardinalité)	len(L)	4
Indexation	L[2]	5
Nbr d'occurrences d'un élément dans L	L.count(5)	1
Sélectionner un slice	L[0 :1]	[1]
	L[1 :]	[2,5,6]
	L[:2]	[1,2]

- Parcourir une liste

Par élément :

```
        for e in L:print(e)
```

Par indice d'élément :

```
    for i in range(len(L)):print(L[i])
```

Parcourir les indices et les éléments :

```
    for i,e in enumerate(L):print(i,"=>",e)
```

- Multiplier une liste [1,2]*3 donne la même liste répétée 3 fois [1,2,1,2,1,2]
- Définir un langage

Remarque 2.6.1 Définir un langage par une liste

Si on définit un langage par une liste, il faut respecter la définition d'un langage en tant qu'un ensemble de mots sans répétition. C'est un critère que la liste ne peut pas assurer et qu'il faut prendre en considération

Considérons la liste L comme un langage (défini par extension). Nous voulons sélectionner un sous-ensemble L_1 de L qui est défini par compréhension : $L_1 = \{x \in L | x \mod 2 = 0\}$ (les nombres paires de L).

```
    x for x in L if x mod 2==0
```

Un autre exemple de langage : Les nombres multiples de 5 et inférieurs à 100

```
x for x in range(101) if x mod 5==0
```

- Pour vérifier si un mot $x \in L$ (défini par extension) :

```
x in L     # boolean result
```

Exercice 1: Quleques manipulations sur les listes

Écrivez une fonction permettant de :

1. Répéter les éléments d'une liste en fonction d'une autre liste. Par exemple si L1=[6,1] et L2=[2,4], le résultat est : [[6,6],[1,1,1,1]]

   Function

   Header

   ```
   def repeat_by_list(L1,L2):
   ```

   Test

   ```
   print(repeat_by_list([6,1],[2,4]))
   ```

2. Réarranger les n éléments d'une liste : 1er élément, n ème élément, 2 ème élément, n-1 ème élément, 3 ème élément, ...

   Function

   Header

   ```
   def symmetric_browse(L):
   ```

   Test

   ```
   print(symmetric_browse([1,3,5,7,9,8,6,4,2]))
   ```

Quelques notions sur: Les chaînes de caractères (mots)

- Une chaîne de caractères est une séquence de caractères, placées entre deux " (ou deux '), ou placées sur plusieurs lignes avec les délimiteurs """.
  Par exemple : w ="abcdef"
- La plupart des opérations sur les listes sont applicables aux chaînes sauf qu'une chaîne de caractères est une liste plutôt immuable. (w[2]= "e" génère une erreur).

> **Remarque 2.6.2 Définir un mot par une chaîne de caractères**
> Les chaînes de caractères vont nous permettre de définir des mots formés par zéro, un ou plusieurs symboles.

- Des opérations basiques permettant de simuler quelques opérations sur les mots.

La longueur d'un mot

```
len(w)  # --> 6

```

Le nombre d'occurrences d'un symbole (caractère) dans un mot

```
w.count("a") # --> 1

```

Concaténation entre deux mots

```
"ab"+"cd" # --> abcd
```

L'exposant w^n

```
w*2 # --> abcdefabcdef
```

Vérifier si un caractère ou une sous-chaîne (un facteur) figure dans un mot :

```
"b" in w # --> True
"ab" in w # --> True
```

Chercher le premier indice de la première (respectivement la dernière) occurrence d'une sous-chaîne :

```
w.find("bc") # --> 1 le premier indice de la 1ere occurrence
w.rfind("d") # --> 3 le premier indice de la derniere occurrence
#Un autre exemple de: rfind()
my_str = "Hello, World, World!"
my_str.rfind("World") # --> 14
```

- Autres opérations :

Joindre les éléments d'une liste pour en faire une seule chaîne. La méthode join() permet de joindre une liste de chaînes en une seule chaîne, en concaténant toutes les chaînes de la liste, avec la chaîne appelante insérée entre chaque élément de la liste.

```
my_list = ["Hello", "World", "!"]
separator = ", "
separator.join(my_list) # --> Hello, World, !
```

Diviser une chaînes de caractères par rapport à un séparateur

```
my_str = "Hello, World!"
my_str.split(",") # --> ['Hello', ' World!']
```

Remplacer toutes les occurrences d'une chaînes (un ou plusieurs caractères) par une autre chaîne :

```
my_str = "Hello, World! Hello, Universe !"
new_str = my_str.replace("Hello", "Hi") # --> Hi, World! Hi, Universe!
```

Formater une chaîne selon un certain format en remplaçant les marqueurs de substitution , par des valeurs fournies en argument, de n'importe quel type (entier, flottant, chaîne, etc.).

```
"{}+{}={}".format(3, 4, 3+4) # --> 3+4=7
```

Il existe une deuxième option avec la syntaxe f-string (disponible à partir de la version Python 3.6) pour formater des chaînes. Les f-strings sont souvent considérés comme plus lisibles et plus faciles à utiliser que la méthode précédente.

```
f"{3}+{4}={3+4}" # --> 3+4=7
```

Récupérer un caractère à partir de son code ASCII ou unicode :

```
chr(66)  # --> B
```

Récupérer le code ASCII ou unicode d'un caractère

```
ord("a")  # --> 97
```