Théorie des Langages (THL)

Dr. O. MESSAOUDI
email: o.messaoudi@univ-bouira.dz

Université Akli Mohand Oulhadj - Bouira
Faculté des Sciences et des Sciences Appliquées
Département d'Informatique
Licence 2
(2019/2020)

Chapitre I
Introduction

1. C'est quoi un langage ?
2. Historiques des langages
3. Langages formels
4. Mots et symboles
5. Langages et grammaires
6. Description d'un langage

C’est quoi un langage ?

Un ***langage*** c’est un support ***naturel*** qui nous permet d’***échanger*** des information et des idées entre nous et également avec la machine.

Un langage naturel est ***informel*** et ***ambigu***, et demande toute la subtilité d’un cerveau humain pour être interprétés correctement.

Une phrase → plusieurs sens

Un langage doit être formalisé et non ambigu pour pouvoir être interprété par la machine.

Une phrase → un seul sens

Historiques des langages

Avant 1940

Ada Lovelace

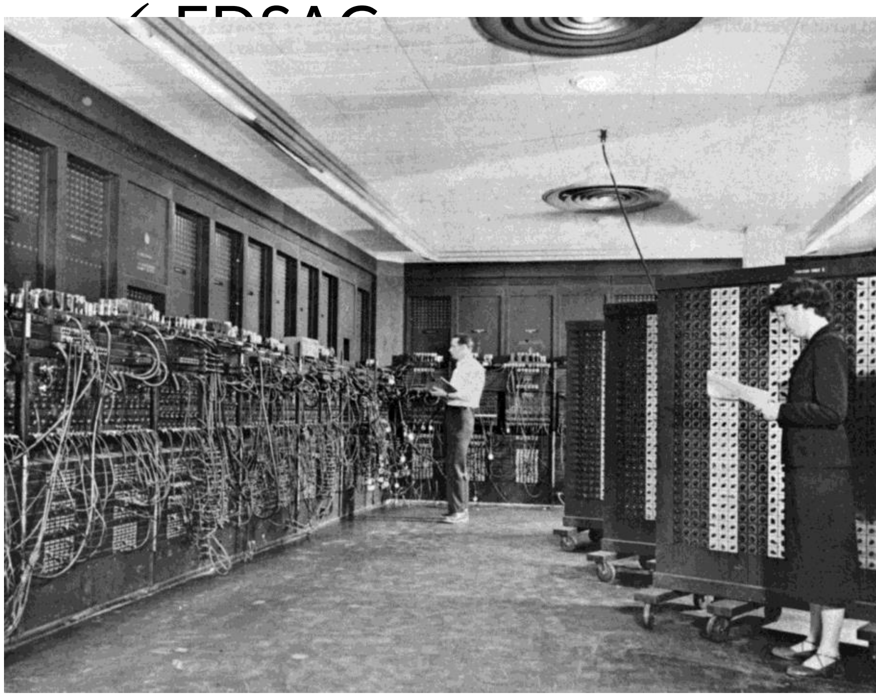
The Countess of Lovelace
1815 – 1852
Known for : Mathematics, computing

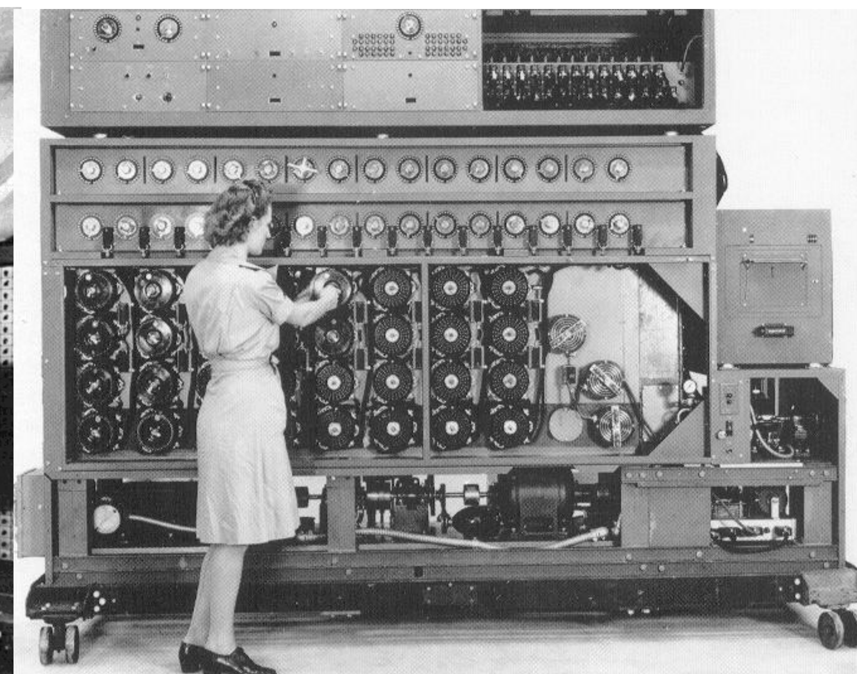
- ❑ mathématicienne et écrivaine,
- ❑ Travaillé sur un ordinateur mécanique, la machine analytique de Charles Baddage,
- ❑ Son rôle : traduction de la mémoire du mathématicien Luigi Menabrea sur la machine analytique,
- ❑ Premier programme et premier algorithme écrit,
- ❑ Langage Ada nommé en son honneur,

Historiques des langages

En 1940

- ❑ Premières machines électroniques,
  - ✓ La Bombe de Turing en Angleterre,
  - ✓ ENIAC, armé américaine, lié à la bombe nucléaire,
- ❑ Premiers ordinateurs électroniques à lampe,
  - ✓ EDSAC

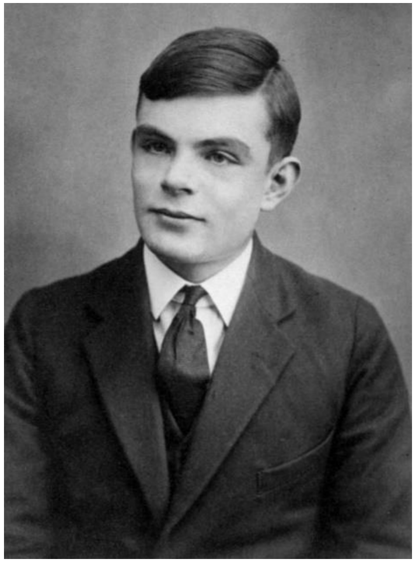


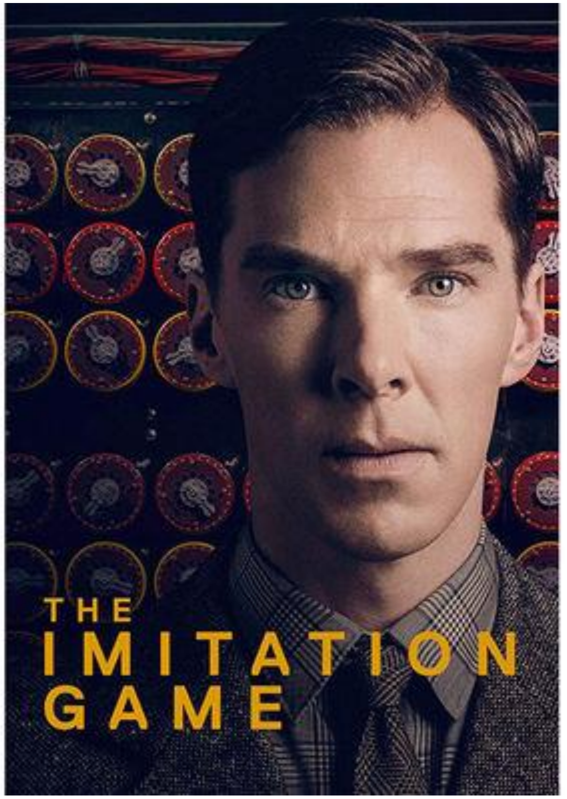


n

Historiques des langages

Alan Michael Turing
1912 - 1954





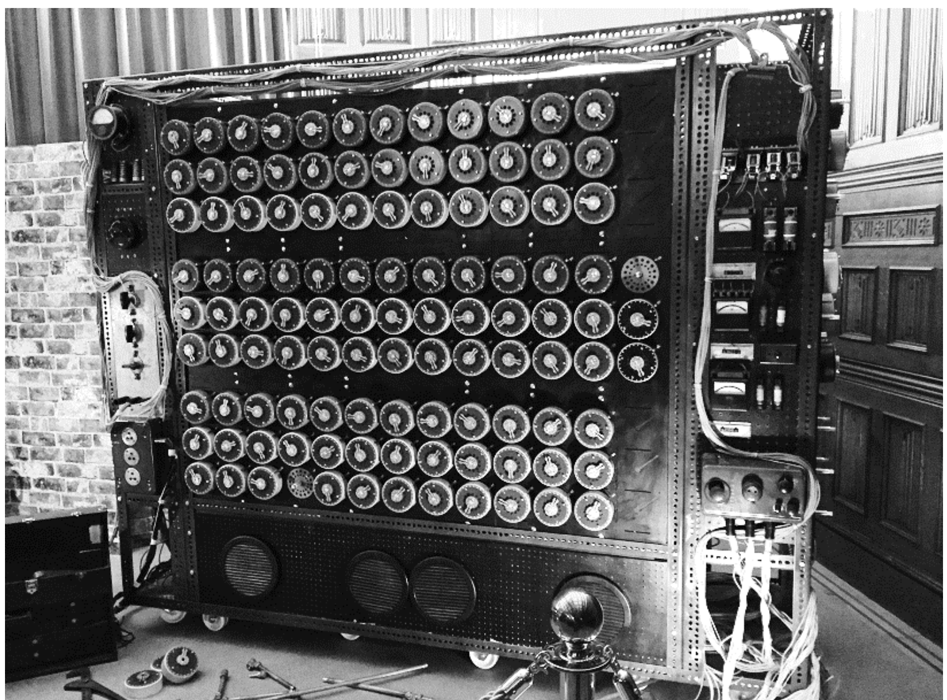
- ❑ Mathématicien Cryptologue,
- ❑ Inventeur du premier ordinateur,
- ❑ Débat sur l'intelligence artificielle : test de Turing
- ❑ Inventeur du concept de *programmation* et de *programme*,

Film à voir : **Imitation Game**

Historiques des langages

Alan Michael Turing

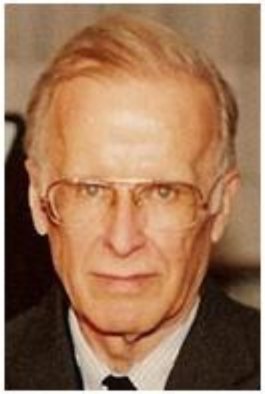
- La bombe (1938, électromécanique)
- Testait automatiquement les codes d'Enigma,
- Casse le code Enigma des Nazis durant la 2ième guerre mondiale,
- Les historiens estiment qu'il a écourté la guerre de 2 ans,
- Reconnu héros de guerre 55 ans après son décès,



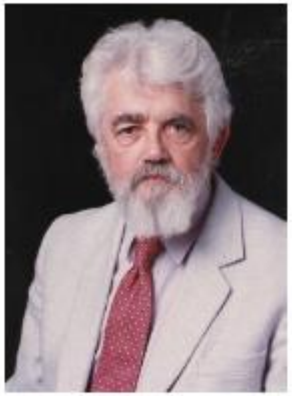
Historiques des langages

1950 - 1960

- FORTRAN (FORmula TRANslator) – John Backus,
- LISP (LISt Processor) premier langage fonctionnel spécialisé dans le traitement des listes – John McCarthy,
- COBOL (Common Business Oriented Language) – Grace Hopper,



John Backus
1924 - 2007



John McCarthy
1927 - 2011

Grace Hopper
1906 - 1984

Historiques des langages

1967 - 1978

- Mise en place des paradigmes fondamentaux,
- *Simula 67* – premier langage Orienté Objet,
- C – premier langage système,
- *Smalltalk* – premier IDE graphique (Objet et Fonctionnel, UI Fenêtrés) – Alan Key,
- *Prolog* – premier langage Logique,
- *ML* – pose les bases de la programmation fonctionnelle, typage statique et polymorphique (basé sur *Lisp*),

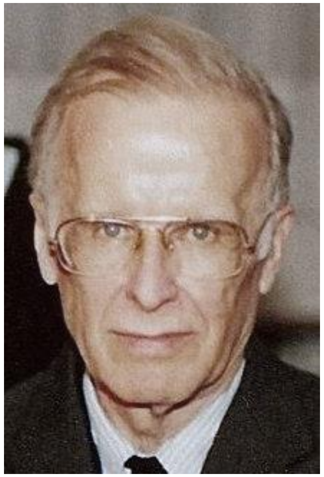
Alan Key
Born 1940

Historiques des langages

1980

- Normalisation et recherche de performance
- Ada, Eiffel, Perl
- C++ - langage système et OO – Bjarn Stroustrup
- FP – Functional Programming – John Backus

Bjarne Stroustrup
Born 1950



John Backus
1924 - 2007

Historiques des langages

1990

- Les langages se tournent vers internet
- Python – Guido van Rossum
- Ruby, Lua, JavaScript, PHP
- Mais aussi : Haskell
- Java – James Gosling
- C# - Hejlesberg (co-inventeur du Pascal)

Anders Hejlsberg
Born 1960

James Gosling
Born 1955

Guido van Rossum
Born 1956

Langages formels

Un langage doit être formalisé et non ambigu pour pouvoir être interprété par la machine. ***Comment ?***

Au départ, un ordinateur ne comprend qu'un seul langage, pour lequel il a été conçu : son ***langage machine***.

Pour communiquer avec des langages plus évolués, il est nécessaire d'utiliser :

- un ***interprète*** qui traduit interactivement les instructions entrées au clavier,
- un ***compilateur*** qui traduit tout un programme,

Langages formels

L'interprétation ou la compilation d'un texte se décompose en trois étapes:

1) ***Analyse lexicale:*** permet de décomposer le texte en entités élémentaires appelées lexèmes (token en anglais),

2) ***Analyse syntaxique:*** permet de reconnaître des combinaisons de lexèmes formant des entités syntaxiques,

3) ***Analyse sémantique:*** permet de générer le code objet directement compréhensible par la machine,

Langages formels

Exemple : le code C suivant

$$cpt = i + 3.14;$$

1) Analyse lexicale permet d'*identifier les lexèmes* suivants :

- ❑ Un **IDENTIFICATEUR** de valeur $\boldsymbol{cpt}$,
- ❑ Un **OPÉRATEUR** de valeur $=$,
- ❑ Un **IDENTIFICATEUR** de valeur $\boldsymbol{i}$,
- ❑ Un **OPÉRATEUR** de valeur $+$,
- ❑ Un **RÉEL** de valeur $\mathbf{3.14}$,
- ❑ Un **POINT VIRGULE**,

Langages formels

Exemple : le code C suivant

$$cpt = i + 3.14;$$

2) ***Analyse syntaxique*** permet de *reconnaître* que cette combinaison de lexèmes forme une *instruction* C *syntaxiquement correcte.*

Cette instruction s'agit d'une affectation entre :

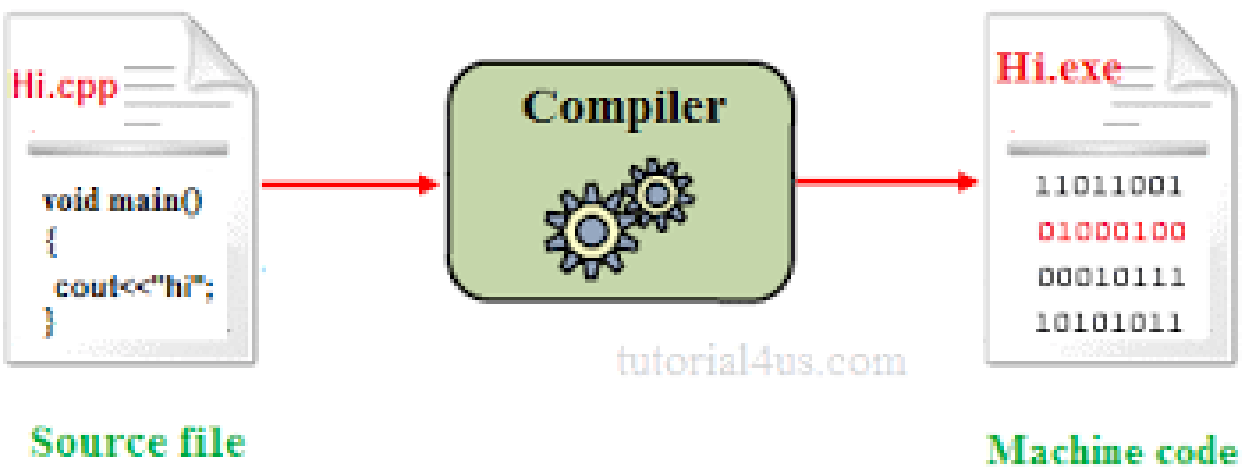
- ❑ La variable d'identificateur $\boldsymbol{cpt}$,
- ❑ Une expression arithmétique résultant de l'addition de la variable d'identificateur $\boldsymbol{i}$ avec le réel $\mathbf{3.14}$,

Langages formels

Exemple : le code C suivant

$$cpt = i + 3.14;$$

3) ***Analyse sémantique*** *vérifie* le bon typage des variables $\boldsymbol{cpt}$ et $\boldsymbol{i}$, puis *génère* le code objet correspondant à cette instruction



Langages formels

En fait, les phases d'analyse lexicale et syntaxique constituent un *même problème* à deux niveaux différents.

Dans les deux problèmes, il s'agit de reconnaître une combinaison valide d'entités :

- ❑ Le cas de l'analyse lexicale : une combinaison de caractères formant des lexèmes,
- ❑ Le cas de l'analyse syntaxique : une combinaison de lexèmes formant des programmes.

La théorie des langages formels permet de résoudre ce type de problème

Alphabets et mots

Un ***alphabet***, noté A, est un ensemble fini non vide de symboles

Exemples d'alphabets :

$$A_1 = \{\circ, \diamond, \star\}$$
$$A_2 = \{a, b, c, \dots, z\}$$
$$A_3 = \{if, then, else, id, nb, =, +\}$$

Un ***mot***, défini sur un alphabet A, est une suite finie d'éléments de A.

Exemples de mots :

- sur l'ensemble A_1, le mot $\circ\circ\star$
- sur l'ensemble A_2, le mot if
- sur l'ensemble A_3, le mot $ifid = nb$

Alphabets et mots

Lors de ***l'analyse lexicale*** d'un programme, l'alphabet est l'ensemble des symboles du clavier, tandis que les mots (ex. les mots clés, les identificateurs, les nombres, les opérateurs, etc.) sont généralement appelés lexèmes.

Lors de ***l'analyse syntaxique*** d'un programme, les éléments de base de l'alphabet sont les lexèmes de l'analyse lexicale, tandis qu'un mot est une suite de lexèmes qui forme un programme.

Alphabets et mots

Longueur d'un mot : La longueur d'un mot u défini sur un alphabet A, notée $|u|$, est le nombre de symboles qui composent u. Par exemple:

$A_1 = \{\circ, \diamond, \star\}$
$A_2 = \{a, b, c, \dots, z\}$
$A_3 = \{if, then, else, id, nb, =, +\}$

- sur l'ensemble A_1, $|\circ\star| = 2$,
- sur l'ensemble A_2, $|if| = 2$,
- sur l'ensemble A_3, $|if id = nb| = 4$,

Le mot vide : le mot vide, noté ϵ, est défini sur tous les alphabets où $|\epsilon| = 0$.

Définition (A^+): on note A^+ l'ensemble des mots de longueur supérieure ou égale à 1 que l'on peut construire à partir de l'alphabet A.

Définition (A^*): on note A^* l'ensemble des mots que l'on peut construire à partir de A, y compris le mot vide :$A^* = A^+ \cup \{\epsilon\}$

Alphabets et mots

Concaténation : Soient deux mots u et v définis sur un alphabet A. La concaténation de u avec v, notée $u.v$ (ou uv), est le mot formé en faisant suivre les symboles de u par les symboles de v.

La concaténation est ***associative*** mais ***non commutative***. La concaténation est une loi de composition interne de A^* et ϵ est son ***élément neutre***. Par conséquent, $(A,.)$ est un ***monoïde***.

Préfixe, suffixe et facteur

- $\boldsymbol{u}$ est un **préfixe** de $\boldsymbol{v}$ si et seulement si $\exists \boldsymbol{w} \in \boldsymbol{A}^*$ tel que $\boldsymbol{uw} = \boldsymbol{v}$;
- $\boldsymbol{u}$ est un **suffixe** de $\boldsymbol{v}$ si et seulement si $\exists \boldsymbol{w} \in \boldsymbol{A}^*$ tel que $\boldsymbol{wu} = \boldsymbol{v}$;
- $\boldsymbol{u}$ est un **facteur** de $\boldsymbol{v}$ si et seulement si $\exists \boldsymbol{w_1} \in \boldsymbol{A}^*$ et $\exists \boldsymbol{w_2} \in \boldsymbol{A}^*$ tels que $\boldsymbol{w_1 u w_2} = \boldsymbol{v}$;

Alphabets et mots

Exemples:

- sur l'alphabet A_2, si $u = aabb$ et $v = cc$, alors $u.v = aabbcc$,
- $u^3 = u.u.u = aabbaabbaabb$ et $|u^3| = |u| \times 3 = 4 \times 3 = 12$,
- sur l'alphabet A_2:
  - a est préfixe de abc parce que $\exists bc \in A^*$ tel que $a.bc = abc$;
  - f est suffixe de def parce que $\exists de \in A^*$ tel que $de.f = def$;
  - b est facteur de abc parce que $\exists a \in A^*$ et $\exists c \in A^*$ tels que $a.b.c = abc$

Série TD N°1: objectifs

- La maîtres des ensembles des alphabets et des mots,
- Opérations sur les alphabets, les mots et les ensembles,
- Definition formelle sur les alphabets et les mots,

Langages et grammaires

Langage : Un langage, défini sur un alphabet A, est un ensemble de mots définis sur A. Autrement dit, un langage est un sous-ensemble de A^*.

Rappel : A^* est l'ensemble de tous les mots qu'on peut construire à partir de A, y compris le mot vide ($A^* = A \cup \{\epsilon\}$).

Deux langages particuliers sont indépendants de l'alphabet A :

- Le langage vide ($\mathcal{L} = \emptyset$).
- Le langage contenant le mot vide ($\mathcal{L} = \{\epsilon\}$).

Exemples: sur l'alphabet $A = \{0,1\}$. Alors, on a:

- $\mathcal{L}_1 = \{0\}$ et $\mathcal{L}_1 \subset A^*$
- $\mathcal{L}_2 = \{1^n / \, n \in N\}$ et $\mathcal{L}_2 \subset A^*$
- $\mathcal{L}_3 = \{01,1,0\}$ et $\mathcal{L}_3 \subset A^*$

Langages et grammaires

Opérations ensemblistes définies sur les langages :

Soient deux langages $\mathcal{L}_1$ et $\mathcal{L}_2$ respectivement définis sur les alphabets A_1 et A_2 :

- L'***union*** de $\mathcal{L}_1$ et $\mathcal{L}_2$ est le langage défini sur $A_1 \cup A_2$ contenant tous les mots qui sont soit contenus dans $\mathcal{L}_1$, soit contenus dans $\mathcal{L}_2$:

$$\mathcal{L}_1 \cup \mathcal{L}_2 = \{u \ / \ u \in \mathcal{L}_1 \ ou \ n \in \mathcal{L}_2\}$$

- L'***intersection*** de $\mathcal{L}_1$ et $\mathcal{L}_2$ est le langage défini sur $A_1 \cap A_2$ contenant tous les mots qui sont contenus à la fois dans $\mathcal{L}_1$ et dans $\mathcal{L}_2$:

$$\mathcal{L}_1 \cap \mathcal{L}_2 = \{u \ / \ u \in \mathcal{L}_1 \ et \ n \in \mathcal{L}_2\}$$

Langages et grammaires

Opérations ensemblistes définies sur les langages :

Soient deux langages $\mathcal{L}_1$ et $\mathcal{L}_2$ respectivement définis sur les alphabets A_1 et A_2 :

- Le ***complément*** de $\mathcal{L}_1$ est le langage défini sur A_1 contenant tous les mots qui ne sont pas dans $\mathcal{L}_1$:

$$C(\mathcal{L}_1) = \{u \setminus u \in A_1^* \; et \; u \notin \mathcal{L}_1\}$$

- La ***différence*** de $\mathcal{L}_1$ et $\mathcal{L}_2$ est un langage défini sur A_1 contenant tous les mots de $\mathcal{L}_1$ qui ne sont pas dans $\mathcal{L}_2$:

$$\mathcal{L}_1 - \mathcal{L}_2 = \{u \setminus u \in \mathcal{L}_1 \; et \; u \notin \mathcal{L}_2\}$$

Langages et grammaires

Produit de deux langages

Le produit ou la concaténation de deux langages $\mathcal{L}_1$ et $\mathcal{L}_2$, respectivement définis sur les alphabets A_1 et A_2, est le langage défini sur $A_1 \cup A_2$ contenant tous les mots formés d'un mot de $\mathcal{L}_1$ suivi d'un mot de $\mathcal{L}_2$:

$$\mathcal{L}_1.\mathcal{L}_2 = \{u.v \backslash\ u \in \mathcal{L}_1\ et\ v \in \mathcal{L}_2\}$$

Le produit de langages est associatif, mais non commutatif.

$(\mathcal{L}_1.\mathcal{L}_2).\mathcal{L}_3 = \mathcal{L}_1.(\mathcal{L}_2.\mathcal{L}_3)$ et $\mathcal{L}_1.\mathcal{L}_2 \neq \mathcal{L}_2.\mathcal{L}_1$

Considérons par exemple les deux langages $\mathcal{L}_1 = \{00,11\}$ *et* $\mathcal{L}_2 = \{0,1,01\}$ *définis sur* $\{0,1\}$.

$$\mathcal{L}_1.\mathcal{L}_2 = \{000,001,0001,110,111,1101\}$$

Langages et grammaires

Puissances d'un langage

Les puissances successives d'un langage $\mathcal{L}_1$ sont définies récursivement par:

- $\mathcal{L}_1^0 = \{\epsilon\}$,
- $\mathcal{L}_1^n = \mathcal{L}_1.\mathcal{L}_1^{n-1}$ pour $n \geq 1$,

Par exemple:

Si $\mathcal{L}_1 = \{00,11\}$, alors $\mathcal{L}_1^2 = \mathcal{L}_1.\mathcal{L}_1^1 = \{0000,0011,1100,1111\}$

Langages et grammaires

Fermeture itérative d'un langage

La fermeture itérative d'un langage $\mathcal{L}$ (ou fermeture de Kleene ou itéré de $\mathcal{L}$) est l'ensemble de mots formés par une concaténation de mots de $\mathcal{L}$:

« $\mathcal{L}^*$ est l'ensemble de mots qu'on peut former à partir de l'ensemble de mots $\mathcal{L}$ »

$$\mathcal{L}^* = \{u \backslash \exists k \geq 0 \ et \ u_1, \ldots, u_k \in \mathcal{L} \ tels \ que \ u = u_1 u_2 \ldots u_k\}$$

Autrement dit, $\mathcal{L}^* = \cup_{i=0}^{\infty} \mathcal{L}^i$
De même, $\mathcal{L}^+ = \cup_{i=1}^{\infty} \mathcal{L}^i$

Langages et grammaires

Description d'un langage

- Un langage fini peut être décrit par l'énumération des mots qui le composent.

- Certains langages infinis peuvent être décrits par l'application d'opérations à des langages plus simples.

- Certains langages infinis peuvent être décrits par un ensemble de règles appelé grammaire.

- Certains langages infinis ne peuvent pas être décrits, ni par l'application d'opérations, ni par un ensemble de règles. On parle alors de langage indécidable.

Langages et grammaires

Grammaires :

- Un langage peut être décrit par un certain nombre de règles. Cette vue du concept de langage a son origine dans des essais de formalisation du langage naturel.

- Le but était de donner une description précise des règles permettant de construire des phrases correctes d'une langue.

Exemple: la grammaire française

Prenons par exemple le sous-ensemble suivant de la grammaire française :

- le vocabulaire est défini par l'ensemble :

$$T = \{le, la, fille, jouet, regarde\}$$

Langages et grammaires

Exemple: la grammaire française

- les catégories syntaxiques sont :
  - la phrase, notée PH,
  - le groupe nominal, noté GN,
  - le verbe, noté V,
  - le déterminant, noté D,
  - Le nom, noté N

- les règles permettant de combiner des éléments du vocabulaire et des catégories syntaxiques pour construire des catégories syntaxiques sont les suivantes :
  - $PH \rightarrow GN\ V\ GN$
  - $GN \rightarrow D\ N$
  - $D \rightarrow le$
  - $D \rightarrow la$
  - $N \rightarrow fille$
  - $N \rightarrow jouet$
  - $V \rightarrow regarde$

  où le symbole → est une abréviation de “peut être composé de”.

- la catégorie syntaxique de départ est la phrase PH.

Langages et grammaires

Exemple: la grammaire française

La phrase “ $la\ fille\ regarde\ le\ jouet$ ” est une phrase correcte pour la grammaire envisagée, comme le montre l’analyse suivante :

$PH \Rightarrow GN\ V\ GN \Rightarrow D\ N\ V\ GN \Rightarrow la\ N\ V\ GN \Rightarrow la\ fille\ V\ GN \Rightarrow la\ fille\ regarde\ GN \Rightarrow la\ fille\ regarde\ D\ N \Rightarrow la\ fille\ regarde\ le\ N \Rightarrow la\ fille\ rgarde\ le\ jouet$

Notons que:

- La grammaire considérée ne prend pas en compte certains aspects du français, comme les accords de genre.
- « $le\ jouet\ regarde\ la\ fille$ » est aussi une phrase syntaxiquement correcte, mais dont la sémantique n’est pas assurée.

Langages et grammaires

Fonctions d'une grammaire

- ❑ Fonctionnement en production: la grammaire indique comment construire des phrases appartenant au langage,

- ❑ Fonctionnement en reconnaissance: la grammaire permet également de décider si une phrase donnée appartient ou non au langage

Dans le cas d'un langage de programmation, on utilise une grammaire pour :

- ❑ Décrire et construire les entités du langage,
- ❑ Savoir si une entité donnée est correcte et appartient au langage,

Langages et grammaires

Grammaire:
Une grammaire est un quadruplet $G = (T, N, S, R)$ tel que

- T est le vocabulaire terminal, c-à-d l'alphabet sur lequel est définit le langage,

- N est le vocabulaire non terminal, c-à-d l'ensemble des symboles qui n'apparaissent pas dans les mots générés, mais qui sont utilisés au cours de la génération.

- R est un ensemble de règles, dites de réécriture ou de production, de la forme :
$$u_1 \rightarrow u_2, \text{avec } u_1 \in (N \cup T)^+ \text{ et } u_2 \in (N \cup T)^*$$

- $S \in N$ est le symbole de départ ou axiome. C'est à partir de ce symbole non terminal que l'on commencera la génération de mots au moyen de règles de la grammaire.

Langages et grammaires

Terminologie:

- Une suite de symboles terminaux et non terminaux (un élément de $(N \cup T)^*$) est appelée une forme.
- Une règle $u_1 \rightarrow u$ telle que $u \in T^*$ est appelée une règle terminale.

Notation:

lorsque plusieurs règles de grammaire ont une même forme en partie gauche, on pourra « factoriser » ces différents règles en séparant les parties droites par des traits verticaux.

Par exemple, l'ensemble de règles $S \rightarrow ab$, $S \rightarrow aSb$, $S \rightarrow c$ pourra s'écrire $S \rightarrow ab \mid aSb \mid c$.

Langages et grammaires

Le langage défini, ou généré, par une grammaire est l'ensemble des mots qui peuvent être obtenus à partie du symbole de départ par l'application des règles de la grammaire.

Comment?

On introduit les notions de dérivation entre formes, d'abord en une étape, ensuite en plusieurs étapes. Enfin, on définit le langage généré par la grammaire comme étant l'ensemble des mots pouvant être dérivés depuis l'axiome.

Langages et grammaires

Dérivation en une étape:

Soient une grammaire $G = (T, N, S, R)$, une forme non vide $u \in (N \cup T)^+$ et une forme éventuellement vide $v \in (N \cup T)^*$.

La grammaire G permet de dériver v de u en une étape (noté $u \Rightarrow v$) si et seulement si :

- $u = xu'y$ (u peut être décomposé en x, u' et y; x et y peuvent être vides),
- $v = xv'y$ (v peut être décomposé en x, v' et y),
- $u' \to v'$ est une règle de R.

Langages et grammaires

Dérivation en plusieurs étapes:

Une forme $v \in (N \cup T)^*$ peut être dérivée d'une forme $u \in (N \cup T)^+$ en plusieurs étapes:

- $u \overset{+}{\Rightarrow} v$: *si* v peut être obtenue de u par une succession de une (1) ou plusieurs dérivation en une étape,

- $u \overset{*}{\Rightarrow} v$: *si* v peut être obtenue de u par une succession de 0, 1 ou plusieurs dérivation en une étape,

Langages et grammaires

Langage généré par une grammaire

Le langage généré par une grammaire $G = (T, N, S, R)$ est l'ensemble de mots sur T qui peuvent être dérivés à partir de S:

$$\mathcal{L}(G) = \{v \in T^* \setminus S \Rightarrow^+ v\}$$

« on obtient le langage $\mathcal{L}$ par une ou plusieurs dérivations de la grammaire G »

Remarques:

- Une grammaire définit un seul langage.
- Par contre, un même langage peut être engendré par plusieurs grammaires différentes

Langages et grammaires

Types de grammaires

La classification des grammaires, définie en 1957 par Noam CHOMSKY, distingue les quatre classes suivantes :

Type 0 : pas de restrictions sur les règles,

$$S \to v$$

Type 1 : grammaires sensibles au contexte ou contextuelles. Les règles de R sont de la forme :

$$uav \to uwv \text{ avec } a \in N, \mathrm{u}, \mathrm{v} \in (N \cup T)^* \; et \; w \in (N \cup T)^+$$

Autrement dit, le symbole non terminal a est remplacé par w si on a les contextes u à gauche et v à droite

Langages et grammaires

Types de grammaires

Type 2 : grammaires hors-contexte. Les règles de R sont de la forme:

$$A \to w \text{ avec } A \in N \ et \ w \in (N \cup T)^*$$

Autrement dit, le membre à gauche de chaque règle est constitué d'un seul symbole non terminal.

Type 3 : grammaire régulière

- À droite, les règles de R sont de la forme :

$$A \to aB \text{ ou } A \to a \text{ avec } A, B \in N \text{ et } a \in T$$

- À gauche, les règles de R sont de la forme:

$$A \to Ba \text{ ou } A \to a \text{ avec } A, B \in N \text{ et } a \in T$$

Autrement dit, le membre de gauche de chaque règle est constitué d'un seul symbole non terminal, et le membre de droite est constitué d'un seul symbole terminal et éventuellement d'un symbole non terminal.

Langages et grammaires

Types de langages

A chaque type de grammaire est associé un type de langage :

- les grammaires de type 3 génèrent les ***langages réguliers***,
- les grammaires de type 2 génèrent les ***langages hors-contexte***,
- les grammaires de type 1 génèrent les ***langages contextuels***,
- les grammaires de type 0 permettent de générer tous les langages "***décidables***".

Ces langages sont ordonnés par inclusion : l'ensemble des langages générés par les grammaires de type n est strictement inclus dans celui des grammaires de type $n-1$ (pour $n \in \{1,2,3\}$).