1 Introduction :

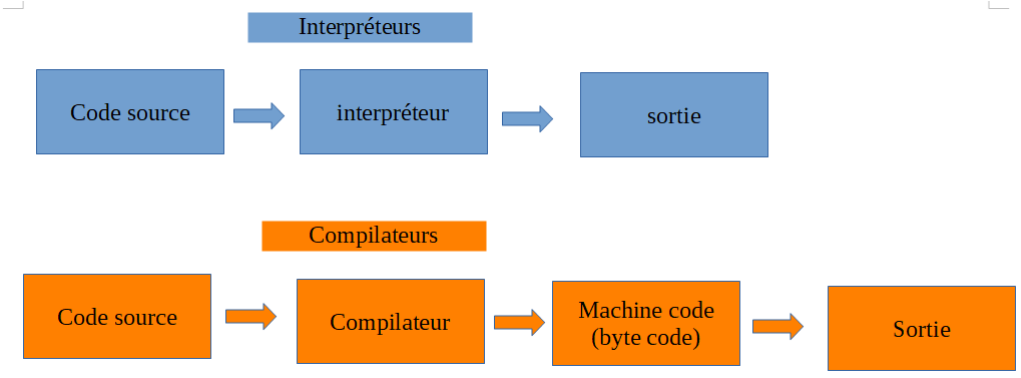
Source : Programme NSI Terminale

Notion de programme en tant que donnée. Calculabilité, décidabilité. Comprendre que tout programme est aussi une donnée.Comprendre que la calculabilité ne dépend pas du langage de programmation utilisé.

Montrer, sans formalisme théorique, que le problème de l'arrêt est indécidable.

L'utilisation d'un interpréteur ou d'un compilateur, le téléchargement de logiciel, le fonctionnement des systèmes d'exploitation permettent de comprendre un programme comme donnée d'un autre programme.

- L'ordinateur à programme enregistré :
  Source : https ://fr.wikipedia.org/wiki/Ordinateur_àprogramme_enregistré
  Les premiers ordinateurs ne pouvaient exécuter qu'un seul programme câblé dans l'ordinateur. Puis les premiers ordinateurs programmables utilisaient des programmes encodés sur des cartes perforées. Enfin, dans les années 40, le concept d'ordinateur à programme enregistré est apparu avec l'idée de construire un ordinateur Turingcomplet (cf chapitre suivant). On retrouve l'équipe de **John Von Neuman** qui travaillait avec ses collaborateurs sur l'ENIAC. **Les programmes sont à présent enregistrés en mémoires comme les données qu'ils prennent en entrée : par des suites de bits (0 ou 1)**. C'est ce concept qui confère la puissance des ordinateurs actuels.
- Interpréteurs Vs compilateurs :
  Source : http ://gallium.inria.fr/ maranget/X/compil/poly/poly001.html
  Les interpréteurs et les compilateurs sont des programmes qui ont pour fonction d'exécuter des programmes ! Ils prennent en entrée un programme sous la forme d'un texte utilisant le format du langage (Java, Python, C, OCaml, Haskell,...). Un interpréteur lit votre programme et l'exécute. Par exemple, python (interprété par l'interpréteur python), javascript (interprété par votre navigateur), bash (interprété par le shell),. . . Chaque instruction de l'utilisateur est immédiatement interprétée et exécutée par l'interpréteur. Un compilateur lit votre programme et le transforme en exécutable. C'est à dire en un programme que la machine peut exécuter par ellemême. On peut le voir comme un traducteur d'un langage compréhensible par l'homme en un langage compréhensible par la machine (bytecode). En général, les programmes compilés s'exécutent plus rapidement grâce à des optimisations effectuées à la compilation. La plupart des langages comme C, Java, Ocaml, Pascal sont compilés. Remarquez que le compilateur Java est écrit en Java !



- Les systèmes d'exploitation :
  Source : https ://interstices.info/a-quoi-sert-un-systeme-dexploitation/
  Les systèmes d'exploitation (partie logicielle des ordinateurs) ont deux rôles :
  — Assurer le stockage des programmes et des données (gestion des fichiers)

— Contrôler l'exécution en gérant les ressources matérielles (mémoire, puissance de calcul) et logiciels (attribuer les ressources aux programmes)

— Assurer la sécurité de l'ordinateur en cas de panne (sauvegarde automatique, redémarrage,...)

Un logiciel d'exploitation est donc un programme qui va exécuter des programmes pour assurer ces trois rôles.

Remarque : revoir le cours11_Os et le TP11_Os de 1ère en NSI

2 Calculabilité - décidabilité - problème de l'arrêt

Y a-t-il des programmes pour tout faire ? Que peut-on calculer ? Que ne pourra-t-on jamais calculer ? C'est à ces questions théoriques que des pionniers de l'informatique comme Alan Turing et Alonso Church ont répondu au milieu des années 1930 en bénéficiant des apports du logicien Kurt Gödel.

Gödel démontra son **théorème d'incomplétude : dans tout axiome mathématique contenant au moins des axiomes arithmétiques, il existe des propositions vraies mais qui sont indémontrables dans la théorie.**

- Calculabilité :

  le mot calcul vient du latin "calculus" qui signifie "caillou"

  La thèse de Church-Turing précise en 1936 que tout programme d'ordinateur, peu importe le langage de programmation, peut être traduit en une machine de Turing.

  Qu'est-ce que la machine de Turing ?

  https ://www.youtube.com/watch ?v=3XG3vZq635A

  Utilisons la machine de Turing sur Exercice 9.1 page 243

  Remarque : la machine de Turing ou les automates d'états finis vont nous révéler le pouvoir d'expression d'un langage.

  Les automates d'états finis :

  Un alphabet est un ensemble de lettres et un langage est un ensemble de mots de cet alphabet.

  Par exemple A={a,b,c} et pour constituer des mots nous allons utiliser 3 opérateurs (*,.,+)
  * : opérateur multiple ; . : opérateur de concaténation et + : opérateur enssembliste.

  ϵ sera l'élément neutre de la concaténation $ab.\epsilon = ab$

  Par exemple $\{a, abc\}^*$ est un ensemble de mots donc un langage constitué par :
  $\{\epsilon\}^* \cup \{a, abc\} \cup \{aa, abcabc, aabc, abca\} \cup \{a, abc\}^3 \cup \{a, abc\}^4 \cup ...$
  $\{a, abc\}^2 = \{aa, abcabc, aabc, abca\}$

  Définition : Un langage est dit régulier s'il peut s'exprimer sous la forme d'une expression régulière (*,.,+)

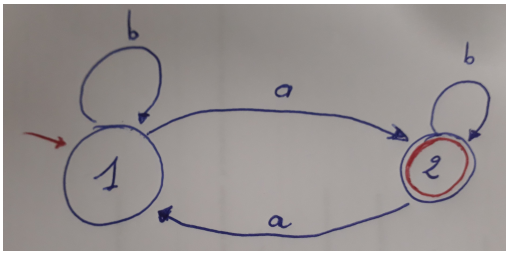
  Théorème : Un langage est régulier si et seulement si il est reconnu par un automate d'états finis.

  Voir la vidéo : automates, des graphes pour décrire des langages

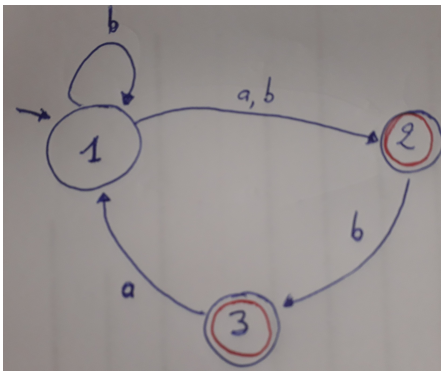
  https ://www.youtube.com/watch ?v=jAUQfvAkHrA

  On va représenter un alphabet A : A={a,b} où les sommets sont les états, les arcs étiquetés par une lettre de l'alphabet.

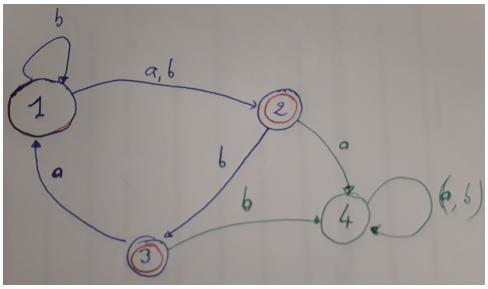
  On notera I : ensemble des états initiaux et T : ensemble des états accepteurs, terminaux.
  Un automate sert à trouver les mots reconnus par le langage : Ex :



Cet automate reconnait l'ensemble des mots avec un nombre impair de a.
On dit qu'un automate est **complet** si pour tout état q et toute lettre de l'alphabet a $\in$ A, il existe un état q' tel que (q,a,q').
On dit qu'un automate est **déterministe** si pour tout état q et toute lettre de l'alphabet, il existe **au plus** un état q' tel que (q,a,q').
Dans notre exemple, l'automate est complet et déterministe.
Prenons un autre exemple :



Cet automate est non déterministe et non complet.
Pour le rendre complet, il faut compléter à la sortie de l'état 2 et l'état 3, voici en vert :



Attention, il faut d'abord rendre un automate déterministe puis on rend l'automate complet.
Méthode pour rendre un automate déterministe :

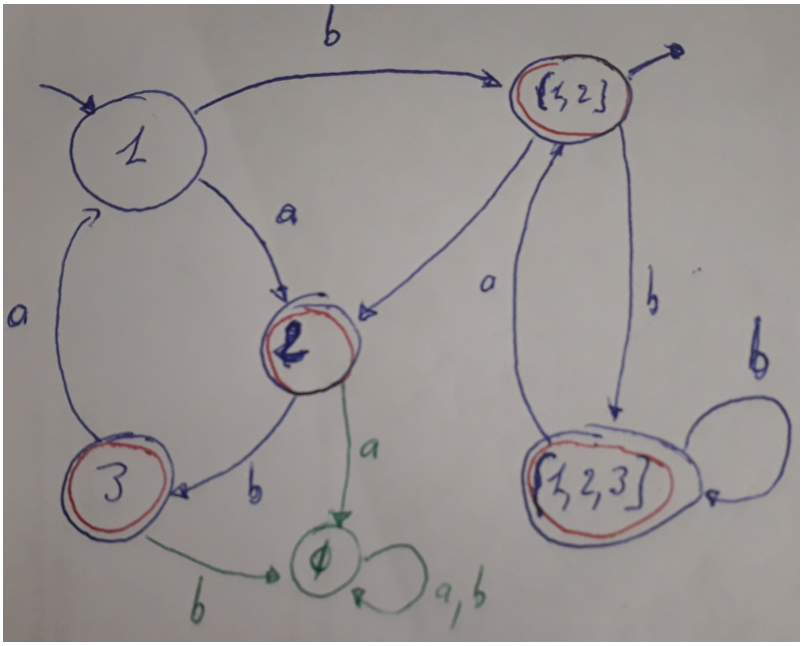
- I = {1} et T = {2,3}
- Construire la table des états :

états	**a**	**b**
1	2	1,2
2	x	3
1	1	x

- L'idée est de reconstruire une table avec un regroupement d'états, on commence par l'union des états initaux, voilà :

états	a	b
{1}	{2}	{1,2}
{2}	x	3
{1,2}	{2}	{1,2,3}
{3}	{1}	x
{1,2,3}	{1,2}	{1,2,3}

- On reconstruit un nouveau graphe déterministe avec I={1} et T= {{2},{1,2},{3},{1,2,3}} et nous traçons le nouveau graphe donnant l'automate déterministe et en vert sa complétude :



Exercices :

A - Soit l'alphabet A= $\{a, b\}$, donner l'automate fini déterministe qui reconnait le langage L =$\{a(b)^*ab\}^*$

B - Donner l'automate fini déterministe et complet pour l'alphabet A= $\{a, b\}$ et le langage L =$\{w \in (a, b)^*/|w|b = 2mod3\}$ où |w|b désigne le nombre d'occurences de b du mot w.

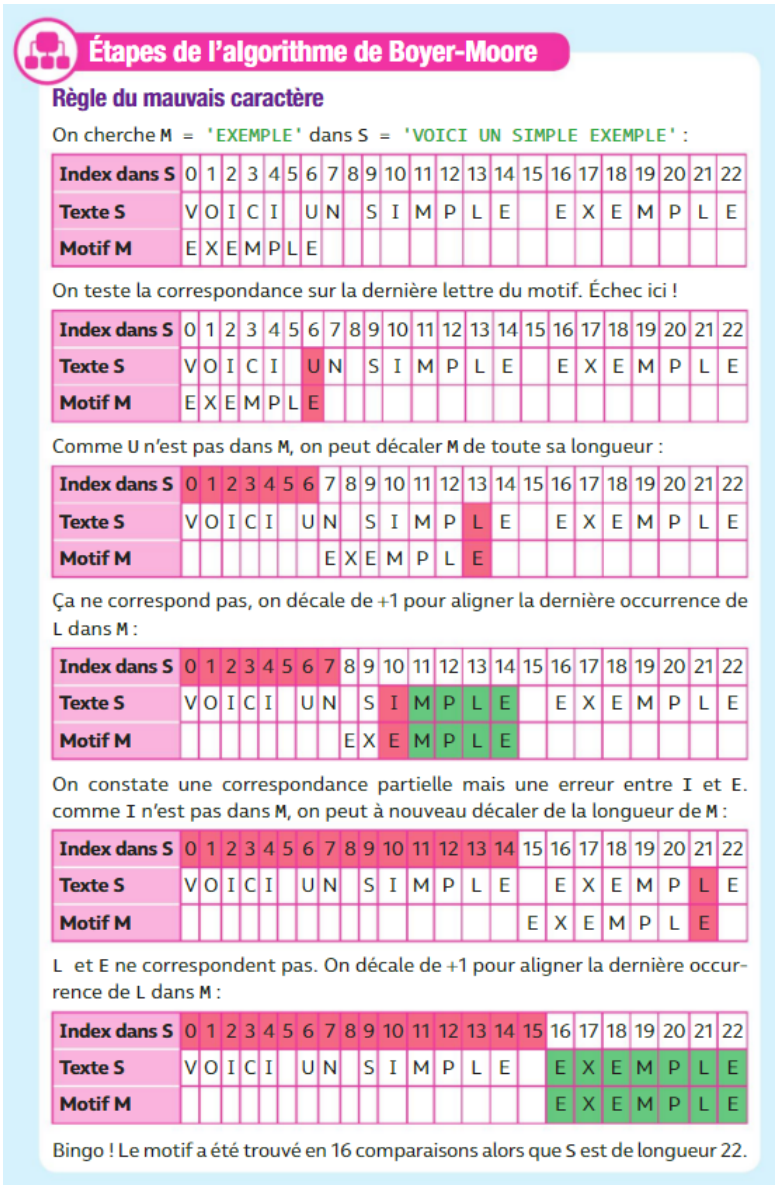
C - Soit un alphabet A= $\{a, b\}$. Donner un automate fini deterministe complet pour le langage L = $\{w \in (a, b)^*$ /w admet comme suffixe bb$\}$.

D - Soit un alphabet A= $\{a, b, c\}$. Construire un automate fini reconnaissant les mots de A^* qui commencent par la lettre a, finissent par la lettre b et contiennent un nombre impair de c.

A quoi cela sert de rendre un automate fini déterministe ? :

Le compilateur ou l'interpréteur doit reconnaître les mots clés du langage par un langage régulier L = $\{(if + while + for + ...)^*\}$

Il y a 2 grandes méthodes de reconnaissance de motif : les automates et l'algorithme de Boyer-Moore. On considère que c'est une branche de l'intelligence artificielle qui fait largement appel aux techniques d'apprentissage automatique et aux statistiques.



Étapes de l'algorithme de Boyer-Moore

Règle du mauvais caractère

On cherche `M = 'EXEMPLE'` dans `S = 'VOICI UN SIMPLE EXEMPLE'` :

Index dans S	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Texte S	V	O	I	C	I		U	N		S	I	M	P	L	E		E	X	E	M	P	L	E
Motif M	E	X	E	M	P	L	E																

On teste la correspondance sur la dernière lettre du motif. Échec ici !

Index dans S	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Texte S	V	O	I	C	I		U	N		S	I	M	P	L	E		E	X	E	M	P	L	E
Motif M	E	X	E	M	P	L	E																

Comme `U` n'est pas dans `M`, on peut décaler `M` de toute sa longueur :

Index dans S	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Texte S	V	O	I	C	I		U	N		S	I	M	P	L	E		E	X	E	M	P	L	E
Motif M								E	X	E	M	P	L	E									

Ça ne correspond pas, on décale de +1 pour aligner la dernière occurrence de `L` dans `M` :

Index dans S	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Texte S	V	O	I	C	I		U	N		S	I	M	P	L	E		E	X	E	M	P	L	E
Motif M									E	X	E	M	P	L	E								

On constate une correspondance partielle mais une erreur entre `I` et `E`. comme `I` n'est pas dans `M`, on peut à nouveau décaler de la longueur de `M` :

Index dans S	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Texte S	V	O	I	C	I		U	N		S	I	M	P	L	E		E	X	E	M	P	L	E
Motif M																E	X	E	M	P	L	E	

`L` et `E` ne correspondent pas. On décale de +1 pour aligner la dernière occurrence de `L` dans `M` :

Index dans S	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Texte S	V	O	I	C	I		U	N		S	I	M	P	L	E		E	X	E	M	P	L	E
Motif M																	E	X	E	M	P	L	E

Bingo ! Le motif a été trouvé en 16 comparaisons alors que `S` est de longueur 22.

Source : Prépabac Spé Nsi Hatier : il faut donc caler le motif et avancer de la gauche vers la droite et comparer de la droite vers la gauche. Le décalage peut se faire suivant 2 méthodes : heuristique du mauvais caractère ou heuristique du mauvais suffixe, en général il est préférable de prendre le max des deux.

Important : Faire un pré-traitement sur le motif pour améliorer la rapidité par rapport à l'algorithme naïf, c'est encore plus efficace sur de gros fichiers.

D'autres algorithmes de recherche textuelle existent comme algorithme de Knuth-Morris-Pratt (KMP).

- **Décidabilité :** une propriété mathématique est dite décidable s'il existe un algorithme, un programme qui détermine en un nombre fini d'étapes si elle est vrai ou fausse. Peut-on décider de manière mécanique (à l'aide d'un programme) si un programme va s'arrêter un jour ? Peut-on décider si deux programmes font exactement la même chose ? Peut-on prouver que tel système d'exploitation ou tel logiciel de pilotage tournera toujours et ne plantera jamais ?

  Alan Turing en 1936 a démontré que que le problème de l'arrêt (halting problem) est indécidable :

  https ://www.youtube.com/watch ?v=PsTcL7KlGBg

Voici un programme python qui illustre le problème de l'arrêt :

Problème de l'arrêt

- Supposons qu'on a la fonction (non-typée) suivante:

```
def arret(fonc, argu):
    """renvoie True si l'appel de fonction fonc avec l'argument argu termine, False sinon."""
```

- On définit alors:

```
def diago(f):
    i = 0
    if arret(f,f):
        while True:
            i = i + 1
    else:
        return i
```

- Que dire de `diago(diago)` ?
  - si `diago(diago)` s'arrête, c'est que `arret(diago,diago)` vaut `False`. Contradiction !
  - si `diago(diago)` ne s'arrête pas, c'est que `arret(diago,diago)` vaut `True`. Contradiction !
  - On a montré par l'absurde, qu'il n'existe pas de fonction `arret`.
- La terminaison d'un programme est indécidable.
  - énormes conséquences pour l'informatique.

SORBONNE UNIVERSITÉ

Conclusion : il n'existe donc pas de programme, d'algorithme qui détermine d'une façon générale si un programme donné se termine sur une entrée donnée.