Avertissement

- Ce cours comporte un survol de notions qui ne sont pas abordées en licence. Certaines affirmations sont imprécises du fait que les définitions correspondantes n'ont pas été données.
  En particulier, la notion de théorie ne sera pas abordée alors que la plupart des propriétés y font référence. Ne pas confondre théorie et système formel (langage + axiomes + règles de déduction).

  - on pourra consulter les articles "histoire de la logique" et "logique mathématique" de l'encyclopédie universelle.

  - revue interstices :

(complexité) : http://interstices.info/jcms/int_63553/non-les-ordinateurs-ne-seront-jamais-tout-puissants

(calculabilité, décidabilité) : http://interstices.info/jcms/c_5723/alan-turing-du-calculable-a-lindecidable

(vidéo) : http://fr.cyclopaedia.net/wiki/Decidabilite-et-indecidabilite

Décidabilité

- Croyance de Hilbert (1862-1943)
  1928 : “ Tout problème mathématique défini doit nécessairement appeler une conclusion : soit une réponse effective à la question posée, soit une preuve de l’impossibilité de sa solution et donc de l’échec inévitable de toute tentative”

  - décidabilité, procédé effectif, algorithme

- le $10^{ème}$ problème de Hilbert (1900) est indécidable (Matijasevic,1970). Décider à l’aide d’une procédure effective si, étant donné un polynôme P à coefficients entiers, il existe des valeurs entières de ses variables telles que P=0 (équation diophantienne). Réponse attendue oui/non. On cherche à résoudre une infinité de cas avec la même procédure.

- bien définir les paramètres du problème : le problème de Hilbert est évidemment décidable si on fixe le polynôme à $x^2 + y^2 - z^2$.

  - $x^2 + y^2 - z^2 = 0$ réponse OUI (3,4,5)
  - $x^4 + y^4 - z^4 = 0$ réponse NON (Théorème de Fermat)

Indécidabilité de la logique du premier ordre

- au programme du cours de LOG licence :

  le calcul des prédicats (logique du premier ordre) muni du système formel D.U.

  est complet (non contradictoire et toute formule vraie *sémantiquement* est démontrable *syntaxiquement*)

  n'est pas décidable (il n'existe pas d'algorithme dont l'entrée soit une formule A quelconque et dont la sortie soit la réponse à la question "A est-elle une tautologie ?").

Il n'existe pas d'algorithme qui ferait automatiquement les démonstrations.

Théorèmes d'incomplétude et d'indécidabilité de Gödel (1931)

- Dans tout système formel non contradictoire assez riche pour contenir une définition des nombres et de l'arithmétique, il existe au moins un énoncé A tel que ni A ni son contraire ne sont démontrables. Ceci affirme l'incomplétude de l'arithmétique. *Cet énoncé A peut être, par exemple, l'affirmation de sa propre indémontrabilité, ou l'affirmation du caractère non contradictoire de l'arithmétique ! (auto référentiel)*

- Il n'existe pas d'algorithme dont l'entrée soit une proposition P quelconque sur les entiers et dont la sortie soit la réponse à la question "P est-elle vraie?". Ceci affirme l'indécidabilité de l'arithmétique.

Problèmes sans algorithmes

- découverts par :

  Alonzo Church,

  Stephen Kleene,

  Emil Post (PCP - 1943),

  Alan Turing (problème de l’arrêt - 1936)

Algorithme ?

- Plusieurs définitions de ce qu'est un algorithme :

  - une suite de règles pour établir des formules mathématiques (Gödel)
  - un ensemble d'instructions pour une machine de Turing
  - un objet du lambda - calcul (Church) :
    un terme est V ou (T T) ou (λV. T)
  - un programme simple (registres en nombre arbitraire, affectation, incrémentation, boucles "pour" imbriquées)
  - théorie des fonctions récursives (Church)

- Toutes ces notions sont équivalentes

  - une machine de Turing peut être simulée par un terme du lambda-calcul
  - les fonctions récursives sont les fonctions calculables par le lambda-calcul
  - etc...

Thèse de Church (1936)

- Opinion : toute nouvelle définition de ce qu'est un algorithme sera forcément équivalente aux précédentes

- avec la "thèse" de Church, l'ensemble des fonctions (partielles et totales) calculables est identique à l'ensemble des fonctions programmables, par exemple en Java

Le problème de l'arrêt : Turing, 1936

- c'est le premier problème connu comme étant indécidable :

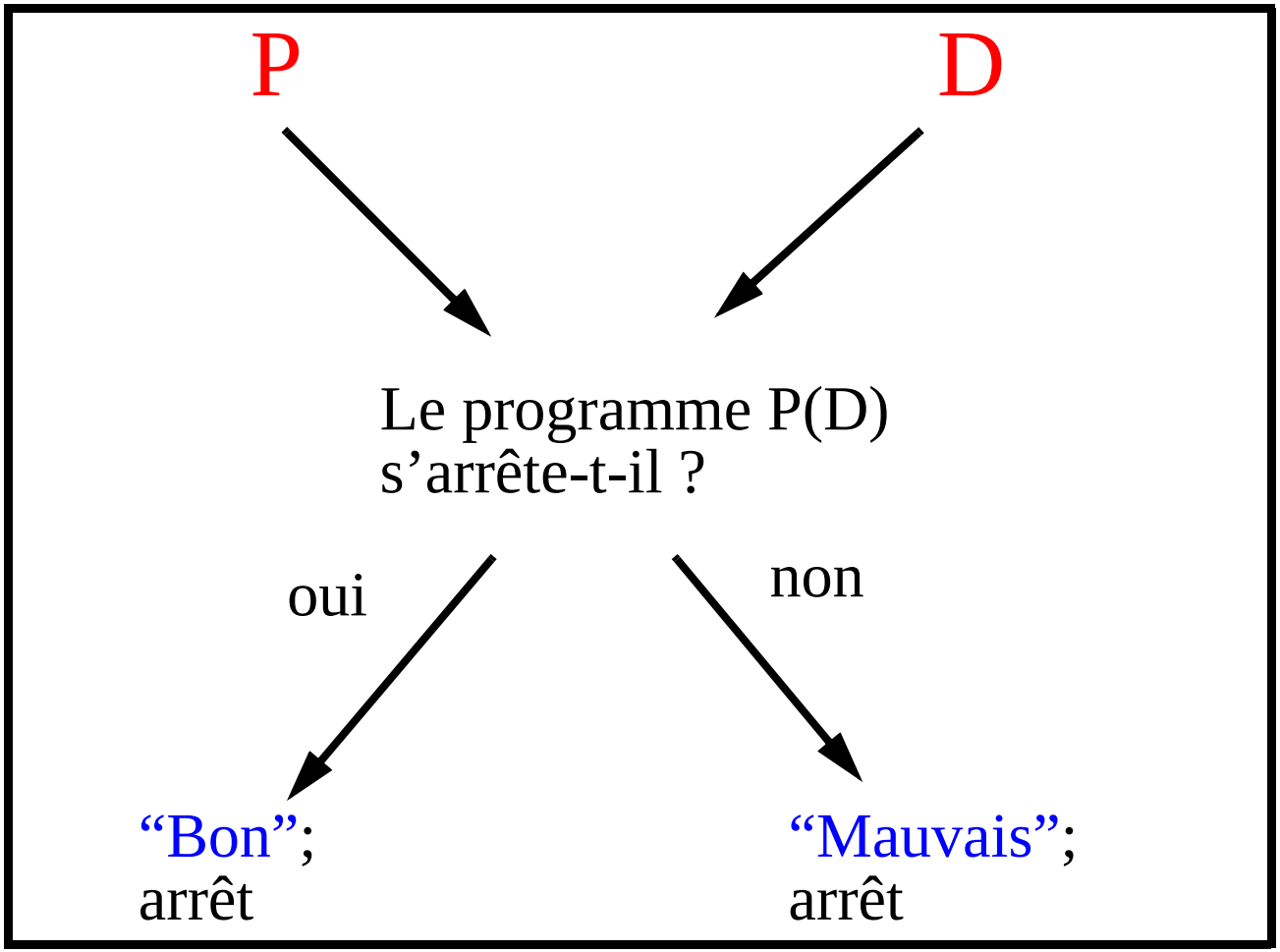
  Existe-t-il un algorithme qui, étant donné le texte d'un programme P, et des données d'entrée D, détermine si P se termine ou pas lorsqu'il est exécuté sur D ?

  L'algorithme recherché, s'il existe, doit toujours répondre OUI ou NON en un temps fini.

- Pour montrer que le problème est indécidable, on va supposer (par l'absurde) qu'un tel algorithme TesterArrêt(P , D) existe et tenter de faire apparaître une contradiction.

- on suppose l'existence de TesterArrêt(P , D)
  Spécif : TesterArrêt décide, au vu de P et D, si P(D) s'arrête ou non. TesterArrêt(P, D) s'arrête pour tous P et D.

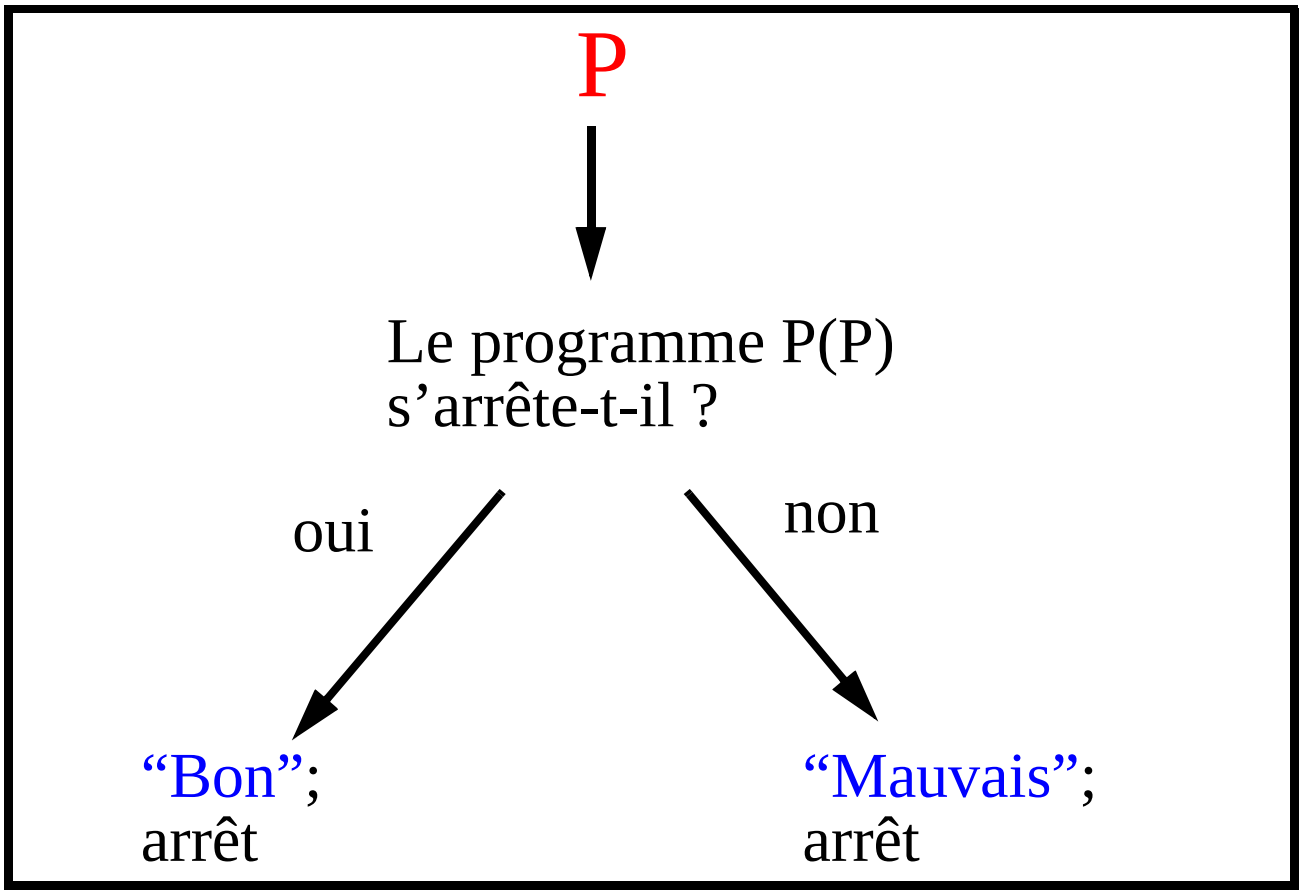
TesterArrêt : **point de vue externe**



on construit...

algo NouveauTesterArrêt(P)
début **point de vue interne**
TesterArrêt (P , P)
fin

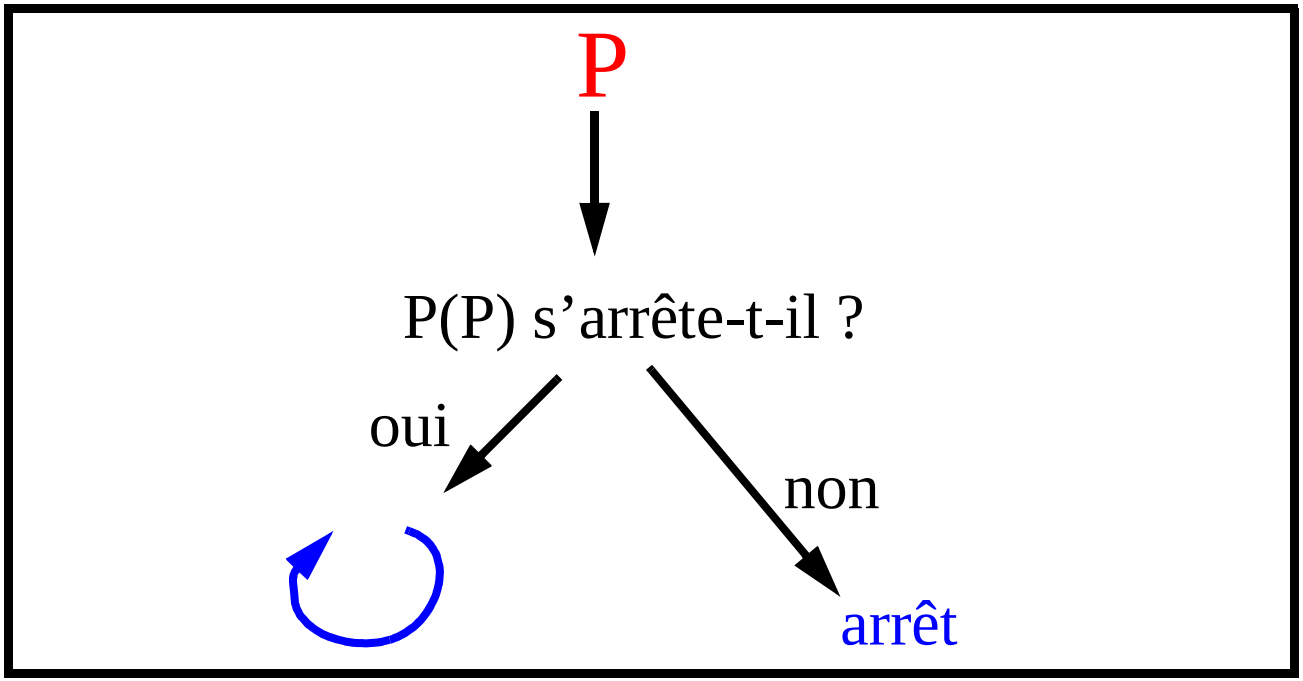
NouveauTesterArrêt : **point de vue externe**



on construit...

algo Drôle (P)
  début **point de vue interne**
    si NouveauTesterArrêt(P) répond “Mauvais”
      alors arrêt
      sinon boucle-sans-fin
    fsi
  fin

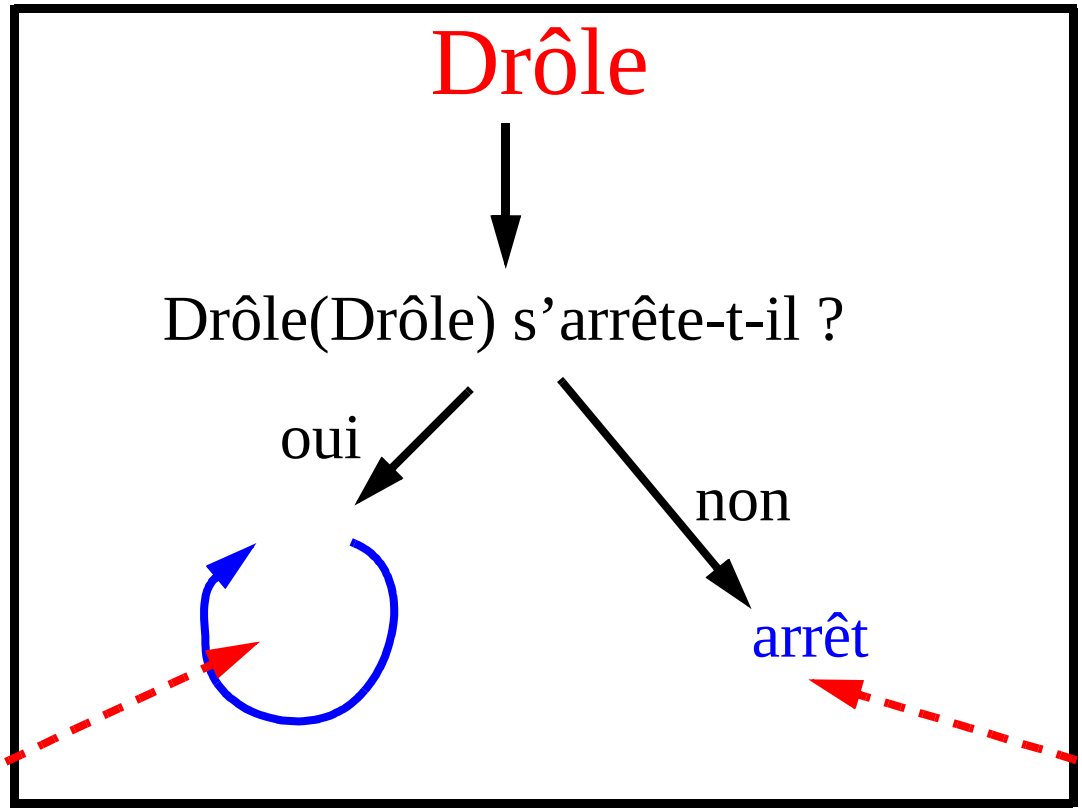
Drôle : **point de vue externe**



Drôle(P)
s’arrête si P(P) boucle,
boucle si P(P) s’arrête.

on exécute Drôle(Drôle)

Drôle(Drôle) : **point de vue externe**



Drôle(Drôle) boucle seulement si Drôle(Drôle) s’arrête

Drôle(Drôle) s’arrête seul[t]si Drôle(Drôle) ne s’arrête pas

Le problème de l’arrêt est indécidable

Drôle ne peut pas exister !

donc TesterArrêt n’existe pas.

Réductibilité

- Une manière (rarement facile !) de prouver qu'un problème P est indécidable est de montrer que, si l'on savait le résoudre, alors on saurait résoudre une problème Q connu pour être indécidable.

  Résoudre Q se réduirait à utiliser l'algorithme pour P sur des données particulières.

  Or Q n'est pas résoluble, donc l'algorithme pour P n'existe pas.

Que faire alors ?

Lorsqu'on a démontré qu'un problème P (qu'on souhaite résoudre) est indécidable, il suit :

- qu'on doit renoncer à le résoudre dans sa forme initiale,
- qu'il faut trouver une version simplifiée de P, puis, soit réussir à la résoudre, soit à nouveau démontrer qu'elle est indécidable et donc la simplifier encore,
- etc...

C'est à dire que l'on cherche des sous-problèmes de P qui sont décidables.

Un problème indécidable en théorie des langages : PCP

- Le problème de correspondance de Post (Emil Post, 1943)

  Étant donné un alphabet X possédant au moins 2 lettres, une instance du problème de correspondance de Post est la donnée d'un entier $k \geq 1$ et de deux suites $u_1, ..., u_k$ et $v_1, ..., v_k$ de chacune k mots sur X.

  On dit que cette instance du problème de Post a une solution s'il existe un entier n et une suite finie non vide $i_1, ..., i_n$ d'entiers dans l'ensemble { 1, 2, ... k } tels que

  $$u_{i_1} u_{i_2} \ldots u_{i_n} = v_{i_1} v_{i_2} \ldots v_{i_n} .$$

  On ne cherche pas la suite, on veut seulement savoir si elle **existe**.

Une instance particulière de PCP

- Soit par exemple l'instance du problème de Post donnée par :

  k = 4

  u1 = aba, u2 = b, u3 = a, u4 = ab,

  v1 = a, v2 = b, v3 = ababa et v4 = b.

  Cette instance du problème de Post admet la solution

  n = 5,

  $i_1 = 1$, $i_2 = 2$, $i_3 = 3$, $i_4 = 2$ et $i_5 = 1$.

  À vérifier !

- PCP est cependant semi-décidable (on trouve un algorithme qui s'arrête si et seulement si la réponse est oui).

Réduction de problèmes en théorie des langages

- Déterminer si l'intersection de deux langages algébriques est vide est un problème indécidable.

  À toute instance de PCP sur X* : k , u_1, ..., u_k , v_1, ..., v_k , on fait correspondre

  - un alphabet A = { a_1, ..., a_k } disjoint de X

  - et deux langages L_1 et L_2 sur $(X \cup A)^*$ tels que :

  PCP a une solution pour la donnée k , u_1, ..., u_k , v_1, ..., v_k si et seulement si L1 $\cap$ L2 est non vide.

- Le problème de correspondance de Post se réduit au problème de la vacuité de l'intersection de 2 langages algébriques.

Ambiguïté

- Déterminer si une grammaire algébrique est ambiguë est un problème indécidable.

  À toute instance de PCP sur X* : k , u_1, ..., u_k , v_1, ..., v_k , on fait correspondre

  - un alphabet A = { a_1, ..., a_k } disjoint de X

  - une grammaire G sur (X ∪ A)* telle que :

  PCP a une solution pour la donnée k , u_1, ..., u_k , v_1, ..., v_k si et seulement si G est ambiguë.

- Le problème de correspondance de Post se réduit au problème de l'ambiguïté des grammaires algébriques.

Problèmes de décision en théorie des langages.

- Dans le tableau qui suit,

  - D signifie que le problème est décidable,

  - V qu'il admet toujours la réponse "oui",

  - I qu'il est indécidable.

  R est un langage reconnaissable.

source : “Théorie des langages et automates”, J.M. Autebert

Problème posé, où L, L_1, L_2 sont de type :	Rat	Det	Alg
L est-il vide ? fini ? infini ?	D	D	D
L = X* ?	D	D	I
$L_1 = L_2$?	D	D 2001	I
$L_1 \subset L_2$?	D	I	I
$L_1 \cap L_2$ est-il vide ? fini ? infini ?	D	I	I
L = R , où R est un rationnel donné ?	D	D	I
L est-il rationnel ?	V	D	I
$L_1 \cup L_2$, $L_1 . L_2$, L_1* sont-ils du même type que L_1 et L_2 ?	V	I	V
$L_1 \cap L_2$ est-il du même type que L_1 et L_2 ?	V	I	I
$L \cap R$ est-il du même type que L ?	V	V	V
$\overline{L}$ est-il du même type que L ?	V	V	I
L est-il non-ambigu ?	V	V	I

Hiérarchie de Chomsky (~1957)

- Une grammaire est un quadruplet

G = < X , V , S , P > où :

- X est un alphabet, dit terminal

- V est un alphabet disjoint de X, dit non-terminal

- $S \in$ V est l'axiome de G

- $P \subset (X \cup V)^*$ **V** $(X \cup V)^* \times (X \cup V)^*$

est l'ensemble (toujours **fini**) des règles de production.

- exemple de règle : a b T a U $\rightarrow$ a W b c d

4 types de grammaires

- type 0 : aucune condition
- type 1 : grammaires contextuelles,
  règles de la forme $u_1 T\, u_2 \rightarrow u_1\, m\, u_2$, $u_1, u_2 \in (X \cup V)^*$, $T \in V$,
  $m \in (X \cup V)^*$, m ne peut pas être le mot vide sauf pour la règle
  $S \rightarrow \varepsilon$, mais dans ce cas S ne figure dans aucun membre droit de règle.
- type 2 : grammaires algébriques,
  $P \subset V \times (X \cup V)^*$
- type 3 : grammaires régulières,
  $P \subset V \times (X\,V \cup X \cup \{\varepsilon\})$

Langage engendré

- comme vu au cours 1 :

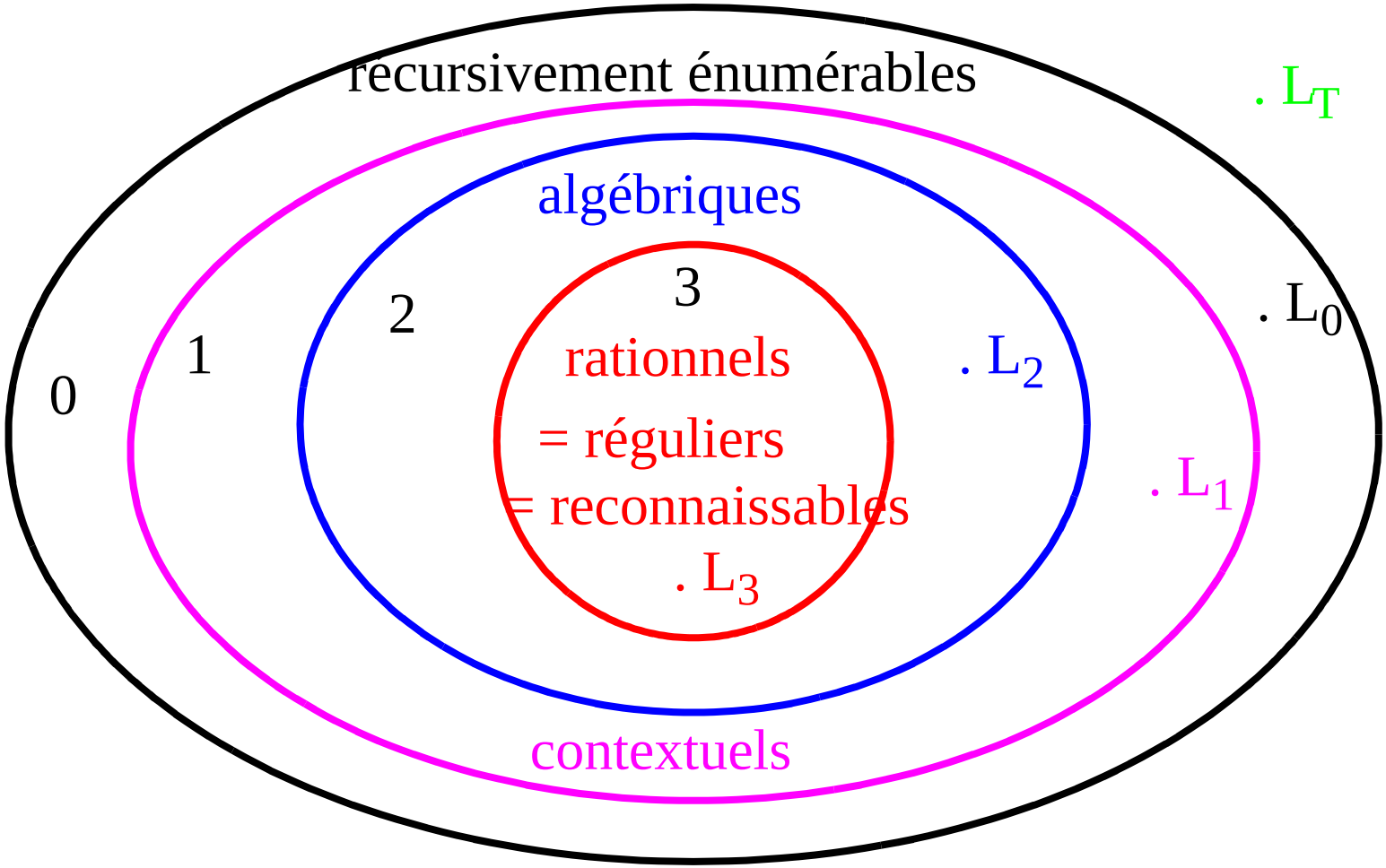
  - on définit $f \rightarrow h$ ssi $f = u\ g\ v$, $h = u\ d\ v$ et $(g, d) \in P$

  - $\rightarrow^*$ est la fermeture réflexive et transitive de $\rightarrow$

  - $L_G(S) = \{ f \in X^* , S \rightarrow^* f \}$

4 types de langages

- Les quatre familles de langages ci-dessous sont strictement incluses les unes dans les autres.



Exemples de langages de types 3, 2, 1 et 0

- $L_3 = \{a^n , n \geq 0 \}$ est reconnaissable
- $L_2 = \{a^n b^n , n \geq 0 \}$ est algébrique et non reconnaissable
- $L_1 = \{a^n b^n c^n, n \geq 0 \}$ n'est pas algébrique (cf cours 9), il est de type 1 comme $\{f.f / f \in \{a, b\}^*\}$ et $\{a^p / p \text{ premier}\}$
- type 0 = langages récursivement énumérables (il existe une machine de Turing qui les accepte). En codant les gram. de type 1 sur {a,b}, les numérotant par ordre hiérarchique et numérotant également les mots de {a,b}* par cet ordre, $L_0 = \{f_i / f_i$ n'est pas engendré par $G_i\}$ est de type 0 mais pas de type 1.
- un langage non récursivement énumérable : on code les machines de Turing sur {a, b} et les numérote : la machine t_i est codée par le mot f_i. On prend $L_T = \{f_i / f_i$ n'est pas accepté par $t_i\}$.

Exemple de grammaire de type 1:

- $G_1 = < X , V , S , P >$ avec :

$X = \{ a , b, c \}$ $V = \{ S , T , W , X , Y , B\}$

$P = \{$ $S \rightarrow abc + aTBc$

$TB \rightarrow TW$

$TW \rightarrow BW$

$BW \rightarrow BT$ ainsi $TB \rightarrow^* BT$

$Tc \rightarrow XBcc$

$BX \rightarrow BY$

$BY \rightarrow XY$

$XY \rightarrow XB$ ainsi $BX \rightarrow^* XB$

$aX \rightarrow aaT + aa$

$B \rightarrow b$ $\}$

Dérivation dans cette grammaire

- $S \rightarrow abc$

  $S \rightarrow aTBc \rightarrow^{3} aBTc \rightarrow aBXBcc \rightarrow^{3} aXBBcc$

  $\rightarrow aaBBcc \rightarrow^{2} aabbcc$

- langage engendré : $\{a^n b^n c^n, n > 0 \}$

  on sait que ce langage n'est pas algébrique

Bibliographie

- Le théorème de Gödel - Kurt Gödel, Ernest Nagel, James R. Newman, Jean-Yves Girard - Seuil 1989 (7,12 euros)

- "La machine de Turing" - Alan Mathison Turing et Jean-Yves Girard - Seuil 1995 (6,60 euros)

- Oncle Petros et la conjecture de Goldbach - Apostolos Doxiadis - Points (6,17 euros)

- Gödel, Escher, Bach. Les Brins d’une Guirlande Eternelle - Douglas Hofstadter - DUNOD 2000 (49 euros)

- Le Théorème du perroquet - Denis Guedj - Points (8,08 euros)

* tarifs juillet 2010

Biographies

- Emil Leon Post (1897 -1954) Mathématicien américain, membre de l'American Mathematical Society depuis 1918 et de l'Association for Symbolic Logic dès sa fondation en 1935. Sa thèse de doctorat, publiée en 1921, porte sur le calcul propositionnel de A. N. Whitehead et B. Russell dont il montre la consistance et le caractère complet. Ici, consistance et complétude sont définies de façon syntaxique. En 1925, poursuivant les travaux de sa thèse, il cherche à montrer le caractère incomplet du système des Principia Mathematica de Russell et Whitehead. Les résultats qu'il obtient ainsi sont contenus dans ceux que K. Gödel et A. Church obtiendront dans les années 1930. Ses principaux travaux sont ensuite consacrés à l'étude des processus effectifs que l'on rencontre en mathématiques. Ce sont ses différents résultats d'indécidabilité qui sont à la base de ceux que l'on rencontre dans la théorie des grammaires formelles. W. Quine écrivait en 1954 à l'occasion de la mort de Post: "Le concept de fonction récursive, concept mathématique précis rendant compte de la notion de calculabilité, fut découvert indépendamment et sous des formes différentes par quatre mathématiciens et Post fut l'un d'entre eux"

- Kurt Gödel (1906-1978). Mathématicien d'origine morave qui, de tout le XXème siècle, a le plus révolutionné les fondements logiques des mathématiques. Gödel montre (dans sa thèse de 1930) la complétude du calcul des prédicats (= logique du premier ordre, qui ne contient pas l'arithmétique) :

tout ce qui est valide (= vrai sémantiquement) est démontrable (syntaxiquement). Sa thèse, et surtout un article publié en 1931 sous le titre "*Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme*" (sur l'indécidabilité formelle des Principia Mathematica et de systèmes équivalents), donneront à Gödel une réputation internationale. Gödel y met fin aux espoirs de Hilbert d'axiomatiser totalement les mathématiques, et de n'en faire qu'une suite de déductions mécaniques ne laissant aucune place à l'intuition.

Gödel montre également qu'il existe des propositions vraies sur les nombres entiers, mais que l'on ne saura jamais démontrer (incomplétude de toute logique contennant l'arithmétique).

- Alonzo Church (1903 - 1995). Mathématicien ayant énoncé sa "thèse" en 1936 à Princeton. La thèse de Church affirme que toute fonction calculable peut l'être avec un ensemble réduit d'instructions. C'est une affirmation philosophique indémontrable qui est la base de toute l'algorithmique... Les travaux de Church ont eu un impact d'importance dans les domaines de l'informatique théorique, de la logique et de la théorie de la récursivité. Il a créé le lamba-calcul dans les années 1930.

- Stephen Kleene (1909 - 1994). Mathématicien ayant soutenu sa thèse (1934) sous la direction de Church, dont les travaux sur le lambda-calcul, les notions de calculabilité et de procédés effectifs de calcul ont étayé la "thèse" de Church. Le théorème de Kleene (Rec = Rat) est l'un des principaux en théorie des langages.

- Alan Turing (1912 - 1954). Mathématicien, logicien et informaticien britannique, qui fut l'un des fondateurs de l'informatique moderne. Il apprend à l'été 1936 les avancées de Max Newman concernant l'élaboration d'une théorie mathématique sur l'incomplétude de Gödel et la question de la décidabilité de Hilbert.1936 est également l'année de la reconnaissance pour Turing ; il reçoit le prix Smith pour ses travaux sur les probabilités, publie "*On Computable Numbers with an application to the Entscheidungsproblem*" et émet le concept de la Machine de Turing. Ce concept constitue la base de toutes les théories sur les automates et plus généralement celle de la théorie de la calculabilité. Il s'agit en fait de formaliser le principe d'algorithme, représenté par une succession d'instructions - agissant en séquence sur des données d'entrée - susceptible de fournir un résultat. Cette formalisation oblige Turing à développer la notion de calculabilité et à déterminer des classes de problèmes décidables. Cela le conduit à introduire une nouvelle classe de fonctions (les "fonctions calculables au sens de Turing") dont il démontre qu'elles sont identiques aux fonctions lambda définissables de Stephen Kleene et Alonzo Church.

- sources :

- http://www.bibmath.net/bios
- http://www.infoscience.fr/histoire/portrait/turing.html
- http://perso.club-internet.fr/vadeker/sciences/post_emil_leon.htm
- encyclopédie universelle